Landscape Image Classification

BCF1
Group 4




Angie Wong (U2121896E)
Keith Heng (U2121807C)
John Lim (U2120457B)

Contents

Problem Statement

In today's context, we are dealing with a vast amount of unstructured image data from cameras and sensors. Image classification is one of the fundamental problems in the field of computer vision.

By using knowledge from SC1015 and Convolutional Neural Networks, we seek to apply image classification on landscape images to identify 6 landscapes, namely Sea, Glacier, Forest, Buildings, Mountain and Street.


sea


glacier


forest


buildings


mountain


street

Approach

1. We chose the Intel Image Dataset which consists of 25k images of natural scenes around the world
2. Data preparation
3. Data cleaning - Balancing & Resizing
4. Data Visualization
5. Apply Convolutional Neural Network model
6. Error Analysis
7. Creating variations of CNN model
8. Comparison of different CNN models

Dataset - Preparation

- Define a function to load image data into folders
- Folders include train and test folders for each type of landscapes
- Study the data set

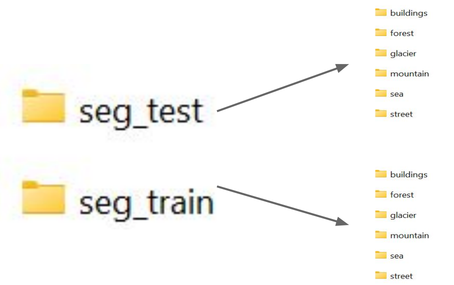
```
(train_images, train_labels), (test_images, test_labels) = load_data()
```

```
Loading C:\Users\ziyan\OneDrive\NTU BCG Y1S2 2022\SC1015 Intro to DSAI\SC1015 project\seg_train
100%|██████████| 2191/2191 [00:02<00:00, 828.80it/s]
100%|██████████| 2271/2271 [00:02<00:00, 806.37it/s]
100%|██████████| 2404/2404 [00:02<00:00, 917.17it/s]
100%|██████████| 2512/2512 [00:02<00:00, 928.24it/s]
100%|██████████| 2274/2274 [00:02<00:00, 908.64it/s]
100%|██████████| 2382/2382 [00:02<00:00, 910.49it/s]
Loading C:\Users\ziyan\OneDrive\NTU BCG Y1S2 2022\SC1015 Intro to DSAI\SC1015 project\seg_test
100%|██████████| 437/437 [00:00<00:00, 564.38it/s]
100%|██████████| 474/474 [00:00<00:00, 555.12it/s]
100%|██████████| 553/553 [00:00<00:00, 745.63it/s]
100%|██████████| 525/525 [00:00<00:00, 695.15it/s]
100%|██████████| 510/510 [00:01<00:00, 419.14it/s]
100%|██████████| 501/501 [00:00<00:00, 805.16it/s]
```

```
train_images, train_labels = shuffle(train_images, train_labels, random_state=25)
```

```python
#Initial dataset
def load_data():
    """
        Load the data:
            - 14,034 images to train the network.
            - 3,000 images to evaluate how accurately the network learned to classify images.
    """
    seg_train = r'C:\Users\ziyan\OneDrive\NTU BCG Y1S2 2022\SC1015 Intro to DSAI\SC1015 project\seg_train'
    seg_test = r"C:\Users\ziyan\OneDrive\NTU BCG Y1S2 2022\SC1015 Intro to DSAI\SC1015 project\seg_test"

    datasets = [seg_train, seg_test]
    output = []

    # Iterate through training and test sets
    for dataset in datasets:

        images = []
        labels = []

        print("Loading {}".format(dataset))

        # Iterate through each folder corresponding to a category
        for folder in os.listdir(dataset):
            label = class_names_label[folder]

            # Iterate through each image in our folder
            for file in tqdm(os.listdir(os.path.join(dataset, folder))):

                # Get the path name of the image
                img_path = os.path.join(os.path.join(dataset, folder), file)

                # Open and resize the img
                image = cv2.imread(img_path)
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                image = cv2.resize(image, IMAGE_SIZE)

                # Append the image and its corresponding label to the output
                images.append(image)
                labels.append(label)

        images = np.array(images, dtype = 'float32')
        labels = np.array(labels, dtype = 'int32')

        output.append((images, labels))

    return output
```

```
(train_images, train_labels), (test_images, test_labels) = load_data()
```



Dataset - Reading Images

Using the opencv-Python library:

- Read the images using imread()

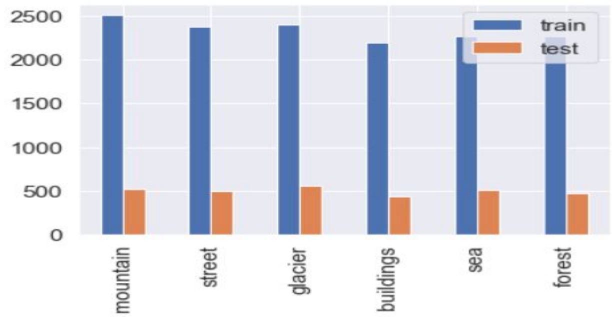
- When we open the images, they are in bgr value, not rgb, so using COLOR_BGR2RGB and cvtColor() we convert it to rgb value

```
# Open and resize the img
image = cv2.imread(img_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
image = cv2.resize(image, IMAGE_SIZE)
```

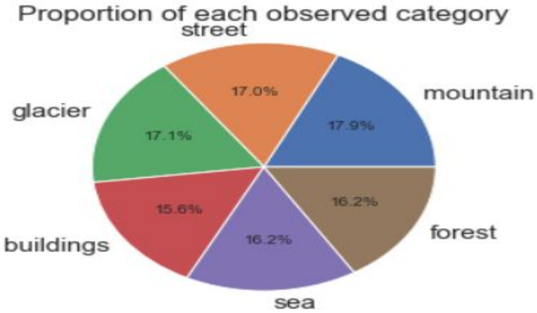
Dataset - Data Visualization

- Study the data set
- The train and test data for each categories are not balanced, hence we need to address it during the cleaning process.
- Overall, it is good that the dataset provided relatively similar portion of each observed category

```python
#Visualise the number of images in train and test dataset
import pandas as pd

_, train_counts = np.unique(train_labels, return_counts=True)
_, test_counts = np.unique(test_labels, return_counts=True)
pd.DataFrame({'train': train_counts,
                    'test': test_counts},
             index=class_names
            ).plot.bar()
plt.show()
```

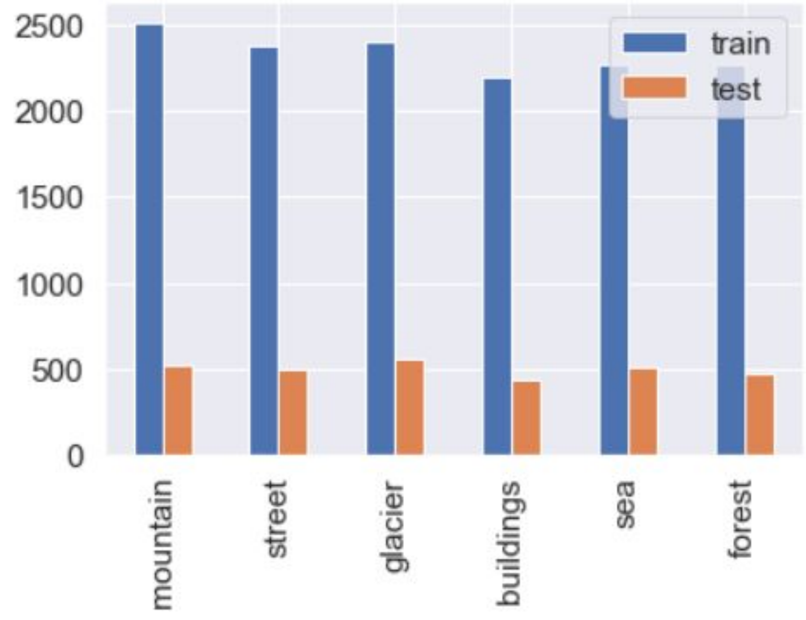



```python
#Visualise the proportion of images based on train data
plt.pie(train_counts,
        explode=(0, 0, 0, 0, 0, 0) ,
        labels=class_names,
        autopct='%1.1f%%')
plt.axis('equal')
plt.title('Proportion of each observed category')
plt.show()
```

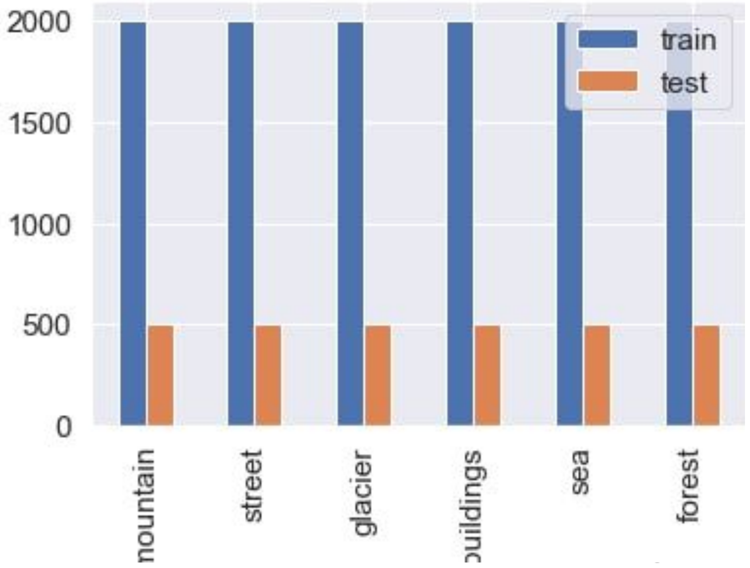


Dataset - Cleaning

We balanced out the number of train data and test data for each type of landscape.



From what we learn in SC1015 it is better to split the train set 80%(2000) and test set 20%(500)



Dataset - Cleaning

We observed there are images of different sizes in the data set, we made some adjustments by resizing the images to 150x150, cleaning the dataset for easier usage.

```python
#Creating labels for the dataset
class_names = ['mountain', 'street', 'glacier', 'buildings', 'sea', 'forest']
class_names_label = {class_name:i for i, class_name in enumerate(class_names)}

nb_classes = len(class_names)

print(class_names_label)

IMAGE_SIZE = (150, 150)
```

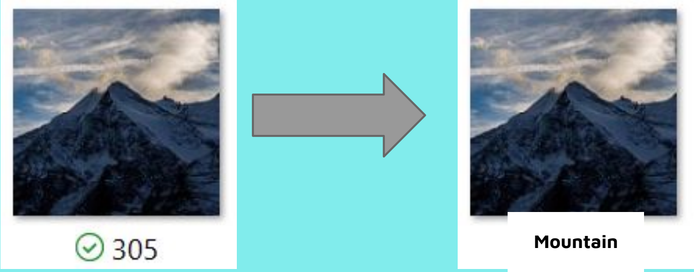
```python
# Open and resize the img
image = cv2.imread(img_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
image = cv2.resize(image, IMAGE_SIZE)
```

random sizes

150 x 150

Dataset - Labeling

Iterate through the folders in the train and test dataset to label the picture with its corresponding class label (Folder) e.g. glacier




```python
#Creating labels for the dataset
class_names = ['mountain', 'street', 'glacier', 'buildings', 'sea', 'forest']
class_names_label = {class_name:i for i, class_name in enumerate(class_names)}
```

```python
# Iterate through each folder corresponding to a category
for folder in os.listdir(dataset):
    label = class_names_label[folder]

    # Iterate through each image in our folder
    for file in tqdm(os.listdir(os.path.join(dataset, folder))):

        # Get the path name of the image
        img_path = os.path.join(os.path.join(dataset, folder), file)

        # Open and resize the img
        image = cv2.imread(img_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        image = cv2.resize(image, IMAGE_SIZE)

        # Append the image and its corresponding label to the output
        images.append(image)
        labels.append(label)

images = np.array(images, dtype = 'float32')
labels = np.array(labels, dtype = 'int32')

output.append((images, labels))
```

Dataset - Visualization

- Scale the data by dividing train_images and test_images by 255 to keep the values between 0 to 1 (as pixel value: 0 to 256)
  - smaller number
  - easier and fast computation

- Define display_random_image with 3 parameters: class_names, images and labels to randomly display an image

```
train_images = train_images / 255.0
test_images = test_images / 255.0
```

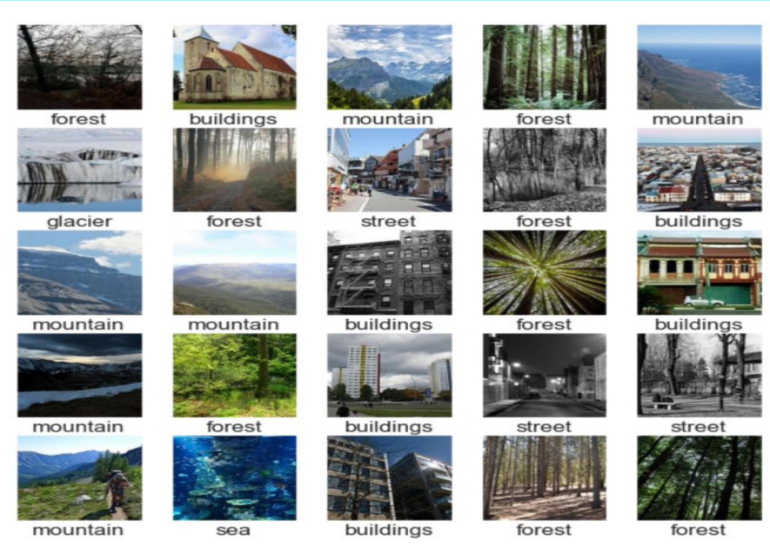
```
def display_random_image(class_names, images, labels):
    """
        Display a random image from the images array and its correspond label from the labels array.
    """

    index = np.random.randint(images.shape[0])
    plt.figure()
    plt.imshow(images[index])
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.title('Image #{} : '.format(index) + class_names[labels[index]])
    plt.show()
```

```
display_random_image(class_names, train_images, train_labels)
```

Image #10806 : forest

Dataset - Visualization

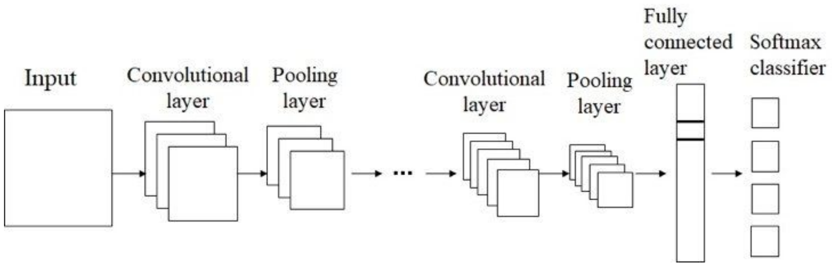
- Extension of the first display function to display 25 images from an array

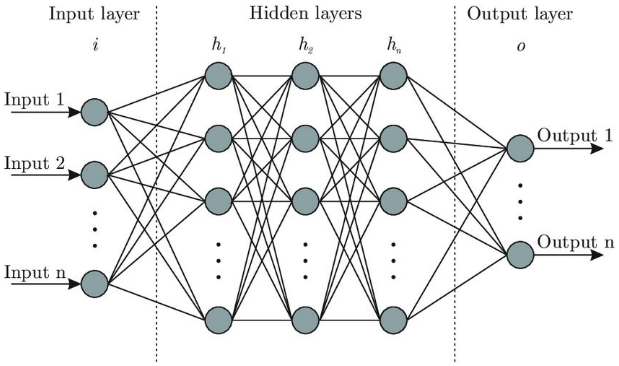



```python
def display_examples(class_names, images, labels):
    """
        Display 25 images from the images array with its corresponding labels
    """

    fig = plt.figure(figsize=(10,10))
    fig.suptitle("Some examples of images of the dataset", fontsize=16)
    for i in range(25):
        plt.subplot(5,5,i+1)
        plt.xticks([])
        plt.yticks([])
        plt.grid(False)
        plt.imshow(images[i], cmap=plt.cm.binary)
        plt.xlabel(class_names[labels[i]])
    plt.show()
```

```python
display_examples(class_names, train_images, train_labels)
```

Why Convolutional Neural Network?





CNN **VS** **ANN**

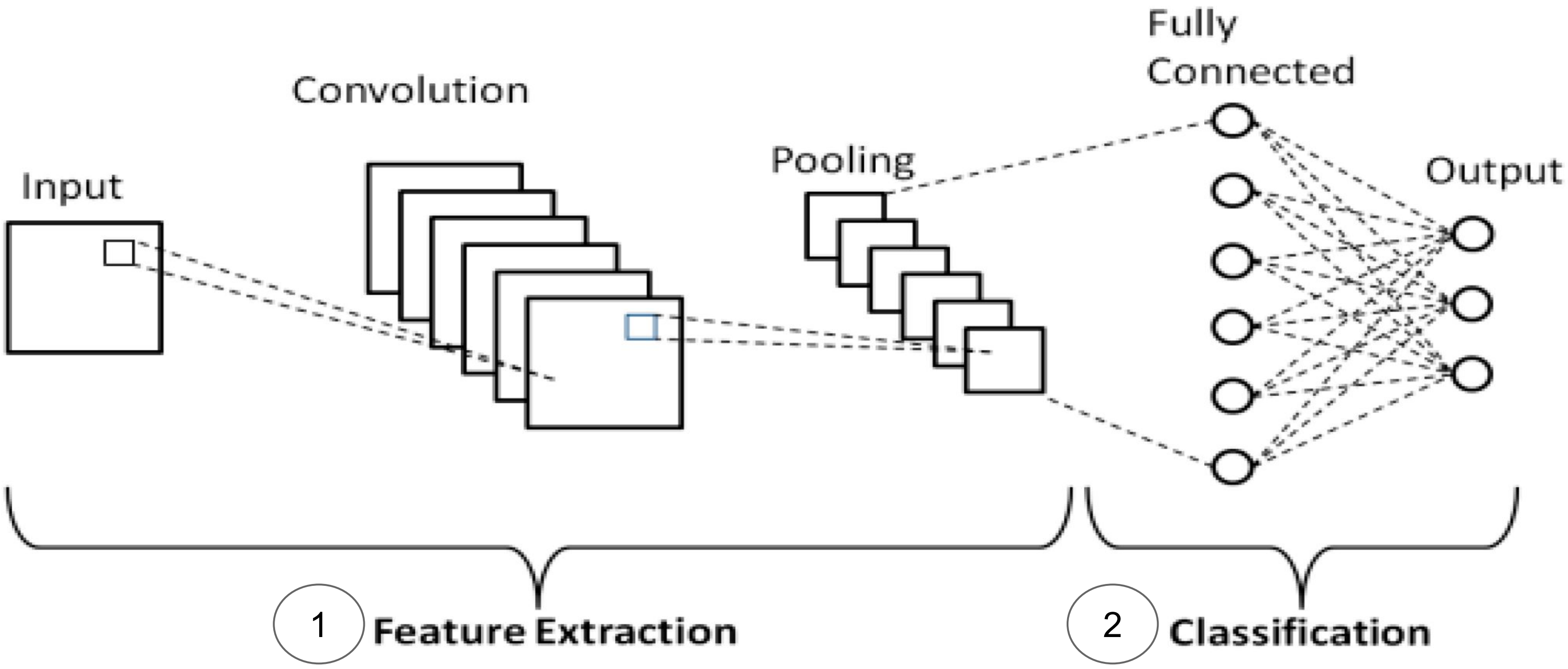
Convolutional Neural Network:

- Less Computationally Intensive
- Extract features to identify and properly classify the image even if the location of the features changes

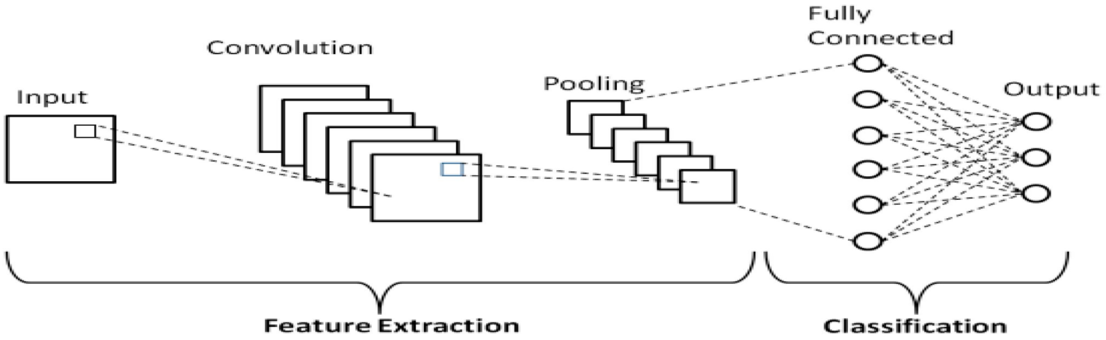
Artificial Neural Network:

- More Computationally Intensive
- Volatile to the changes of features in the image

What is Convolutional Neural Network?



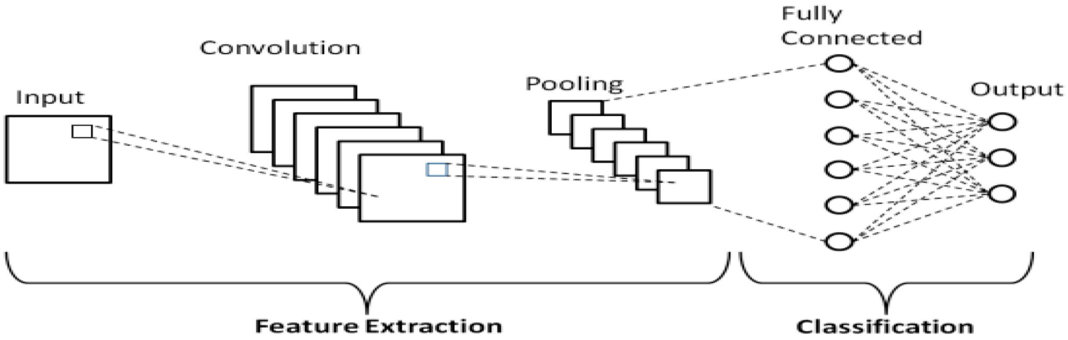
CNN Model 1



- In feature extraction:
  - Use 32 3x3 filters with no padding (Convolutional layer)
  - Activation function: ReLu
    - Non-linear activation function to set all negative values in matrix to 0

```
#Baseline model1
model1 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu', input_shape = (150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(6, activation=tf.nn.softmax)
])
model1.summary()
```

CNN Model 1

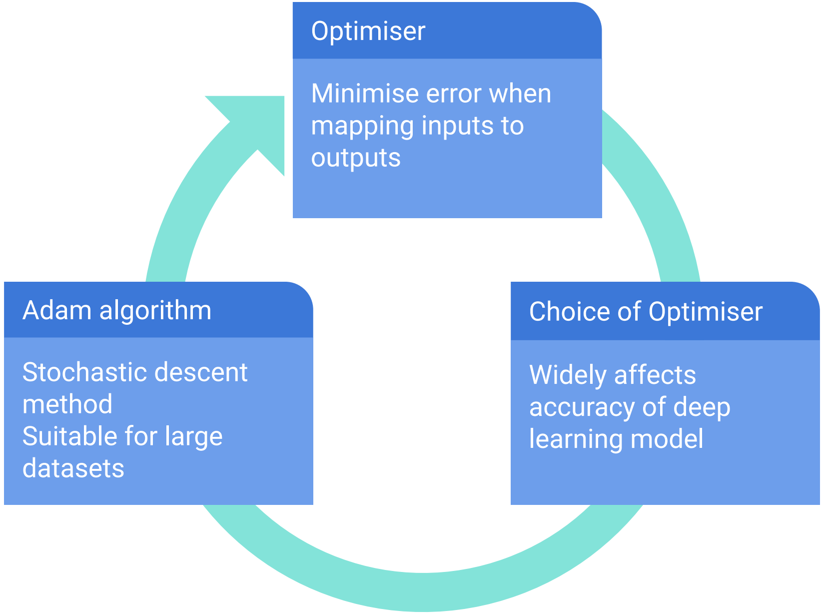


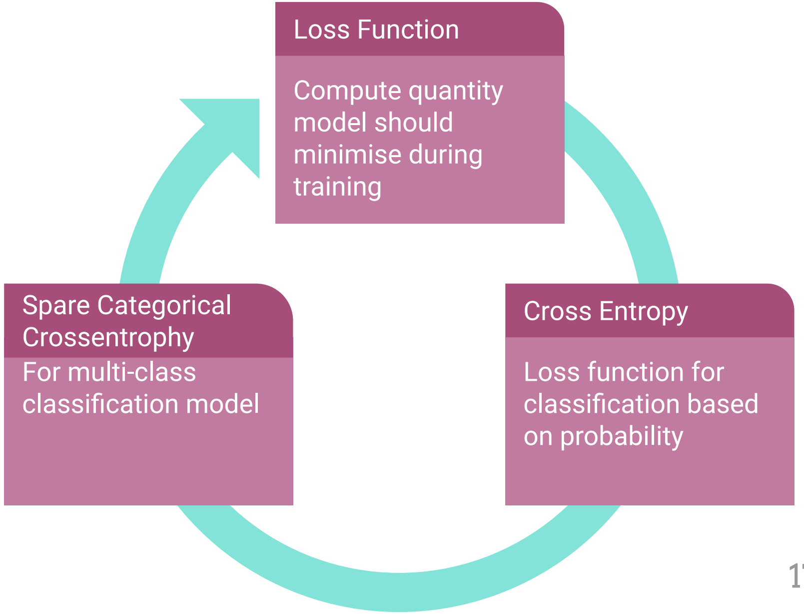
- In feature extraction:
  - Put feature map into 2x2 max pooling layer
  - After convolution blocks, do flattening to convert multidimensional to single dimension.

```
#Baseline model1
model1 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu', input_shape = (150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(6, activation=tf.nn.softmax)
])
model1.summary()
```

Compiling Model

```python
# compiling model with appropriate optimiser and loss functions
model1.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics=['accuracy'])
```





Model fitting

```
# fit the model & include validation split
history1 = model1.fit(train_images, train_labels, batch_size=128, epochs=20, validation_split = 0.2)
```

Validation Split

- Validation set is a data set separate from training set
- Used to validate model performance during training

Validation Process

- Gives info that helps us tune the model
- Model trained on training set
- Model evaluation performed on validation set after every epoch

Outcome

- Prevent overfitting
- Model is really good at classifying samples in training set
- But cannot generalise and classify accurately on new datasets

Accuracy & Loss of Training of Neural Network

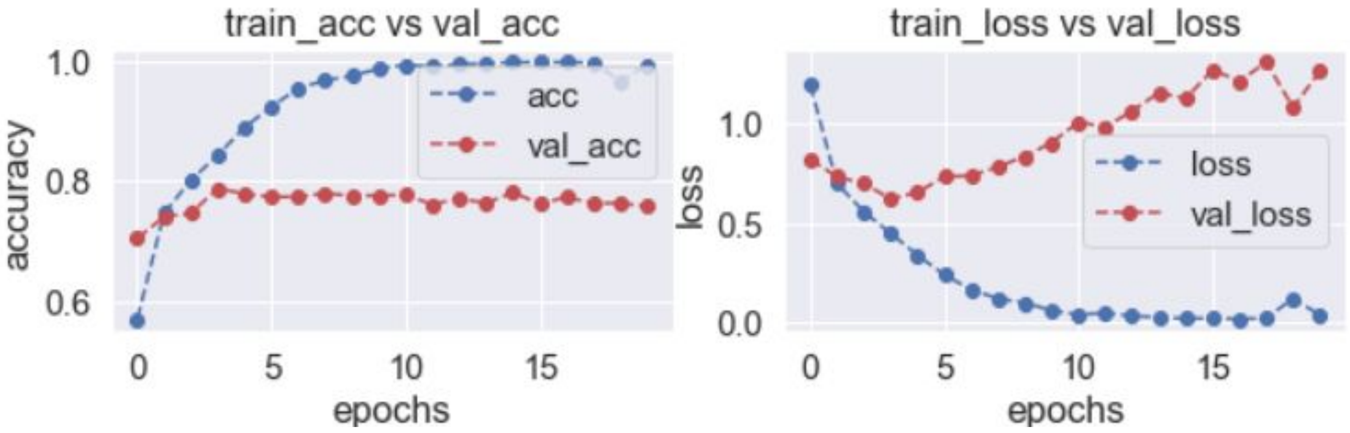
```python
def plot_accuracy_loss(history):
    """
        Plot the accuracy and the loss during the training of the nn.
    """
    fig = plt.figure(figsize=(10,5))

    # Plot accuracy
    plt.subplot(221)
    plt.plot(history.history['accuracy'],'bo--', label = "acc")
    plt.plot(history.history['val_accuracy'], 'ro--', label = "val_acc")
    plt.title("train_acc vs val_acc")
    plt.ylabel("accuracy")
    plt.xlabel("epochs")
    plt.legend()

    # Plot loss function
    plt.subplot(222)
    plt.plot(history.history['loss'],'bo--', label = "loss")
    plt.plot(history.history['val_loss'], 'ro--', label = "val_loss")
    plt.title("train_loss vs val_loss")
    plt.ylabel("loss")
    plt.xlabel("epochs")

    plt.legend()
    plt.show()
```

	Low Loss	High Loss
Low Accuracy	A lot of small errors	A lot of big errors
High Accuracy	A few small errors	A few big errors

$$\text{Cross-entropy} = -\sum_{i=1}^{n}\sum_{j=1}^{m} y_{i,j}\log(p_{i,j})$$

$$Accuracy = \frac{\text{No of correct predictions}}{\text{Total no of predictions}}$$

Train Accuracy/Loss vs Validation Accuracy/Loss




```
94/94 [==============================] - 6s 63ms/step - loss: 1.2161 - accuracy: 0.7463
```

Loss value decreasing in training set

+

Loss value not decreasing in validation set

→

Overfitting has occurred

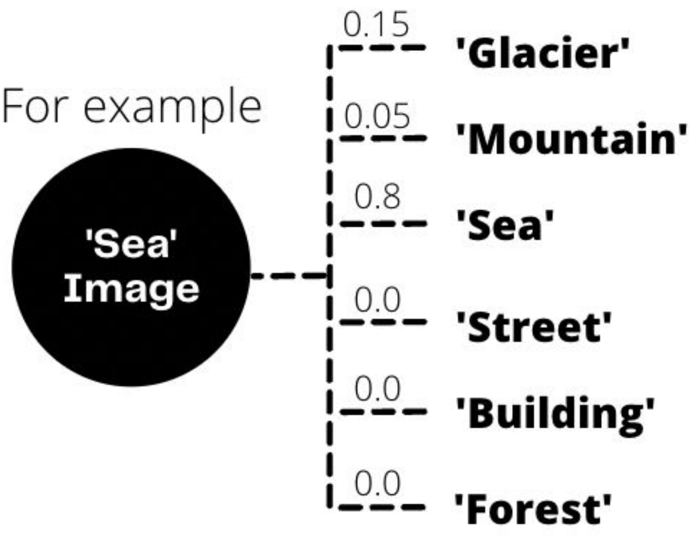
Overlearning from training examples, poor ability to generalise & predict for new data sets

Simple Visualisation of Model's Classifications

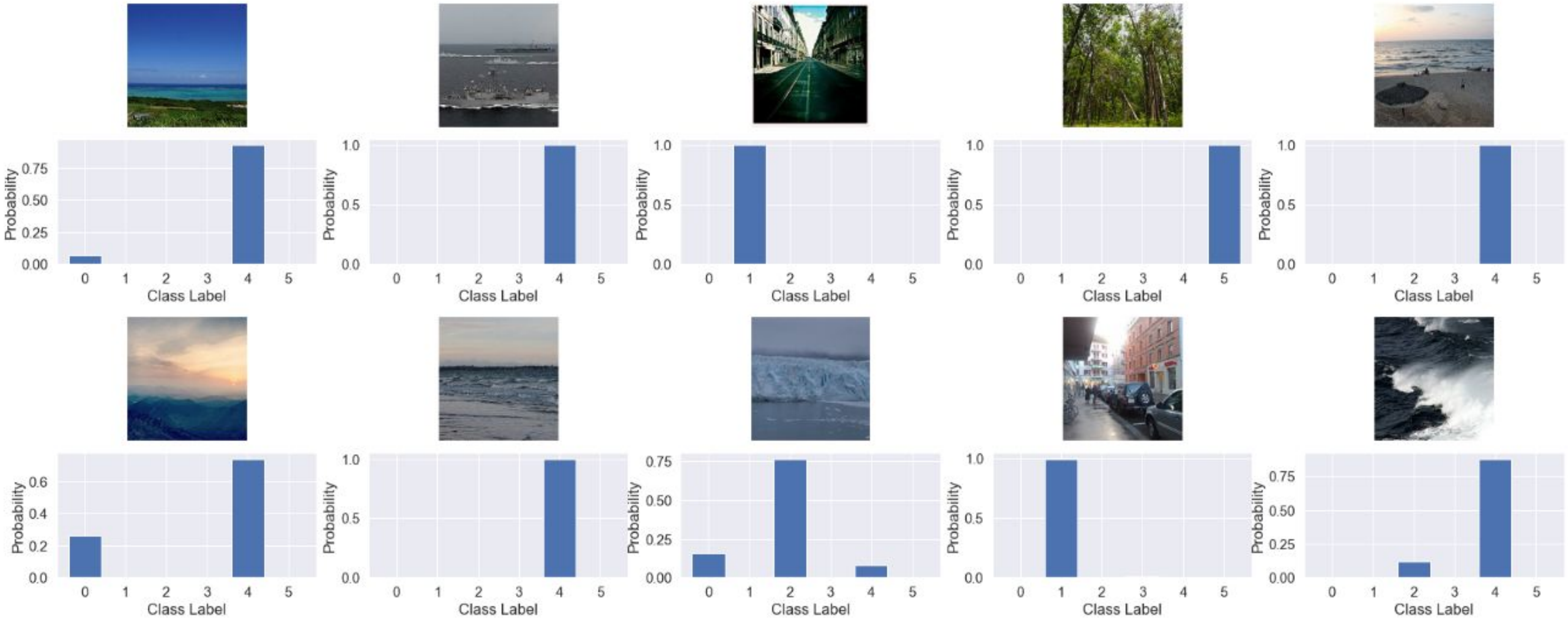
```
# Function to visualise probabilities of image being classified into various categories

fig = plt.figure(figsize=(30, 30))
outer = gridspec.GridSpec(5, 5, wspace=0.2, hspace=0.2)

for i in range(10):
    inner = gridspec.GridSpecFromSubplotSpec(2, 1,subplot_spec=outer[i], wspace=0.1, hspace=0.1)
    rnd_number = randint(0,len(test_images))
    pred_image = np.array([test_images[rnd_number]])
    pred_class = np.argmax(pred_image, axis = 1)
    pred_prob = model1.predict(pred_image).reshape(6)
    for j in range(2):
        if (j%2) == 0:
            ax = plt.Subplot(fig, inner[j])
            ax.imshow(pred_image[0])
            #ax.set_title(pred_class[0])
            ax.set_xticks([])
            ax.set_yticks([])
            fig.add_subplot(ax)

        else:
            ax = plt.Subplot(fig, inner[j])
            ax.bar([0,1,2,3,4,5],pred_prob)
            fig.add_subplot(ax)
            ax.set_xticks([0,1,2,3,4,5])
            # Set common labels
            ax.set_xlabel('Class Label')
            ax.set_ylabel('Probability')

plt.show()
```

Function to visualise the probability of an image being classified into each category

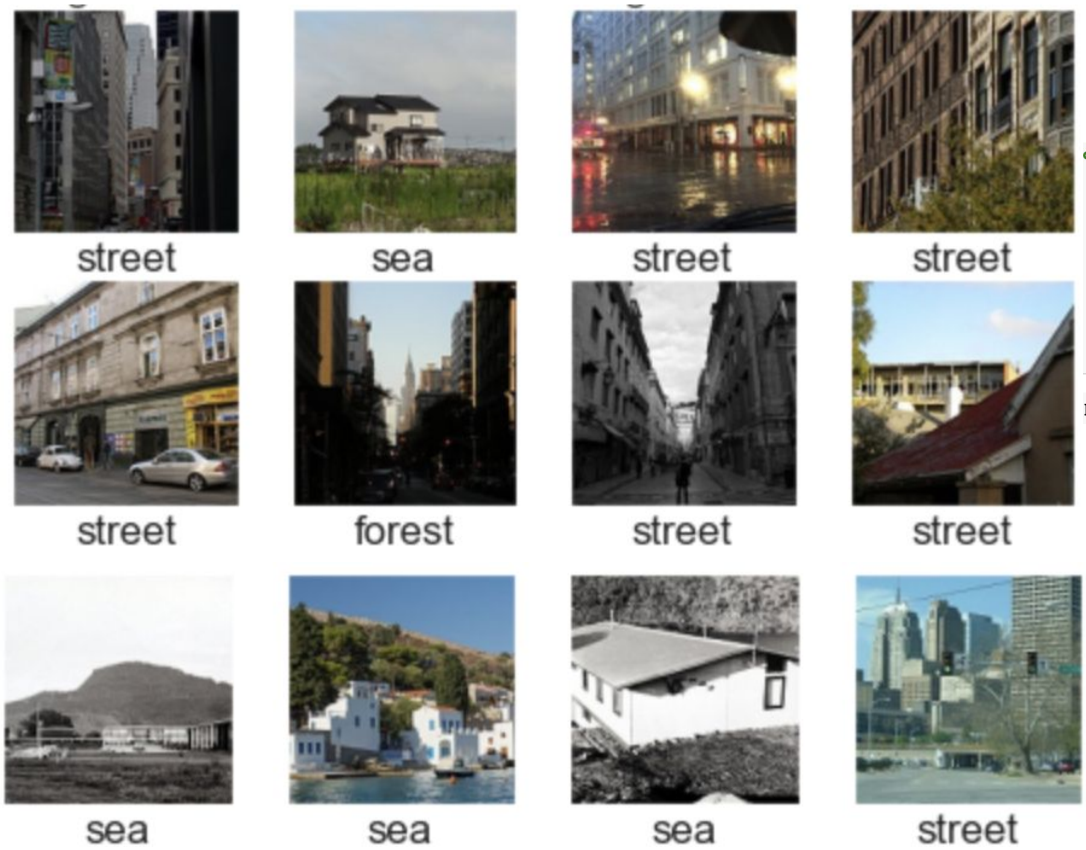
For example



Simple Visualisation of Model’s Classifications



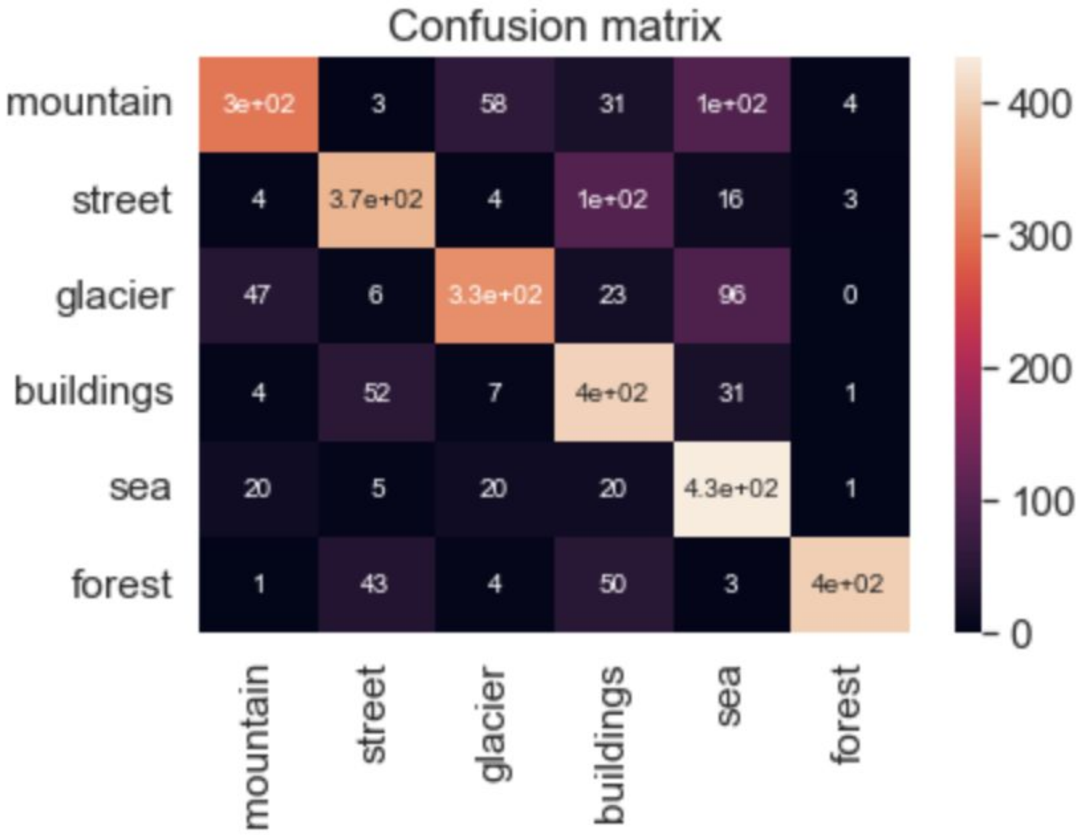
Simple Visualisation of Misclassified Images




```python
def print_mislabeled_images(class_names, test_images, test_labels, pred_labels):
    """
        Show 25 random images wrongly labelled by classifier & their wrong labels
    """
    BOO = (test_labels == pred_labels)
    mislabeled_indices = np.where(BOO == 0)
    mislabeled_images = test_images[mislabeled_indices]
    mislabeled_labels = pred_labels[mislabeled_indices]

    title = "Some examples of mislabeled images by the classifier:"
    display_examples(class_names,  mislabeled_images, mislabeled_labels)
```

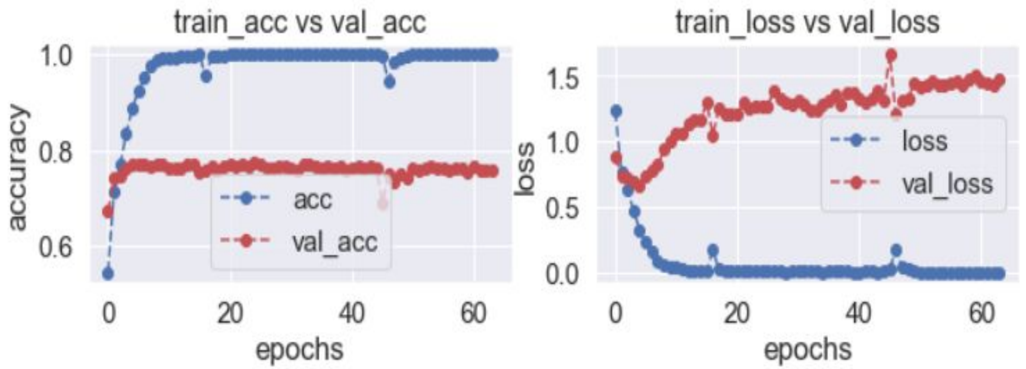
```python
print_mislabeled_images(class_names, test_images, test_labels, pred_labels)
```

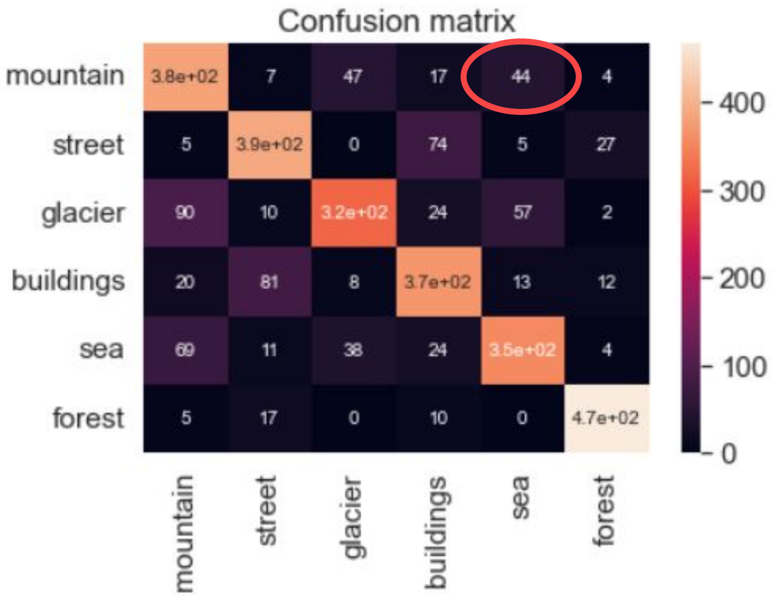
Confusion Matrix of Model's Classifications



CNN Model 1 - Increasing from 20 to 64 epochs

```
plot_accuracy_loss(history1)
```



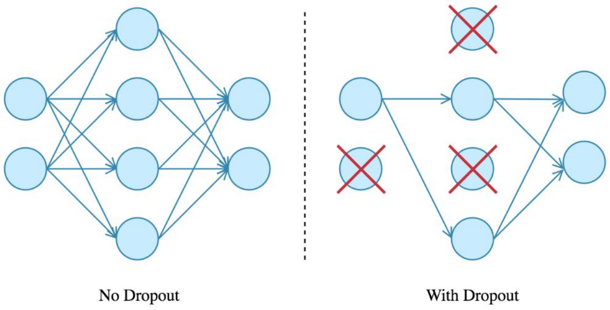



```
test_loss = model1.evaluate(test_images, test_labels)
```

```
94/94 [==============================] - 6s 61ms/step - loss: 1.5125 - accuracy: 0.7583
```

CNN Model 2 - Introducing Dropout Layers

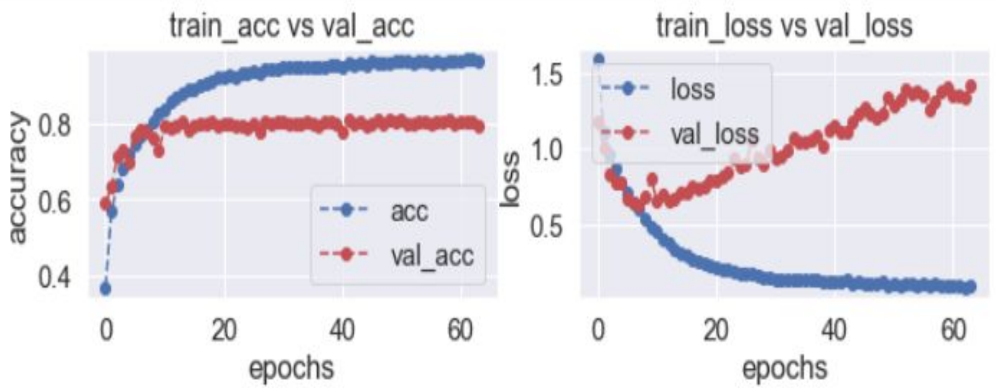
```
model2 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu', input_shape = (150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.25),

    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.25),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(6, activation=tf.nn.softmax)
])
model2.summary()
```

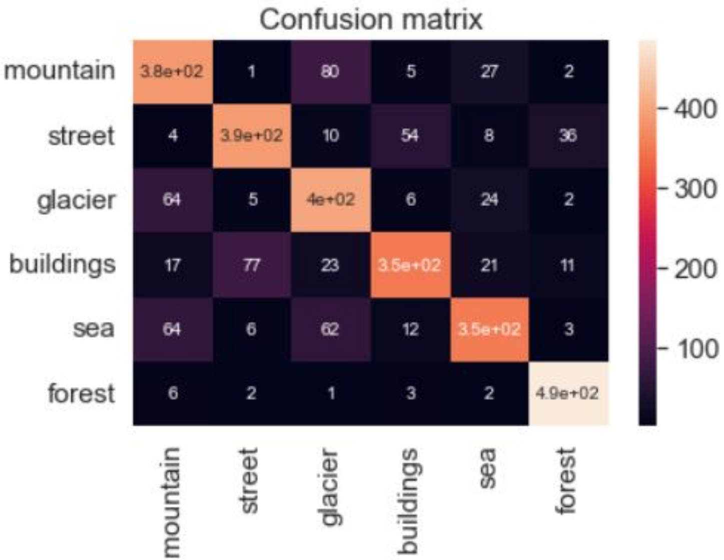


- Reduce overfitting
- Model learn more general and robust patterns from the data

CNN Model 2 - Introducing Dropout Layers

```
plot_accuracy_loss(history2)
```





	mountain	street	glacier	buildings	sea	forest
mountain	3.8e+02	1	80	5	27	2
street	4	3.9e+02	10	54	8	36
glacier	64	5	4e+02	6	24	2
buildings	17	77	23	3.5e+02	21	11
sea	64	6	62	12	3.5e+02	3
forest	6	2	1	3	2	4.9e+02

```
test_loss = model2.evaluate(test_images, test_labels)
```

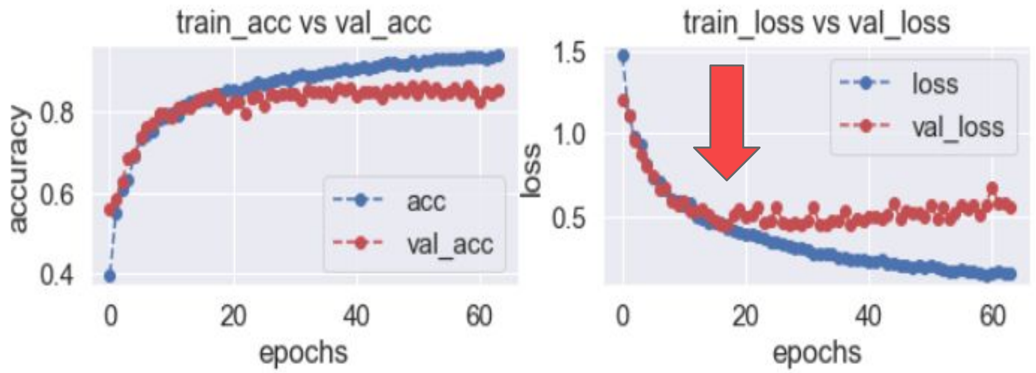
```
94/94 [==============================] - 6s 65ms/step - loss: 1.3706 - accuracy: 0.7873
```

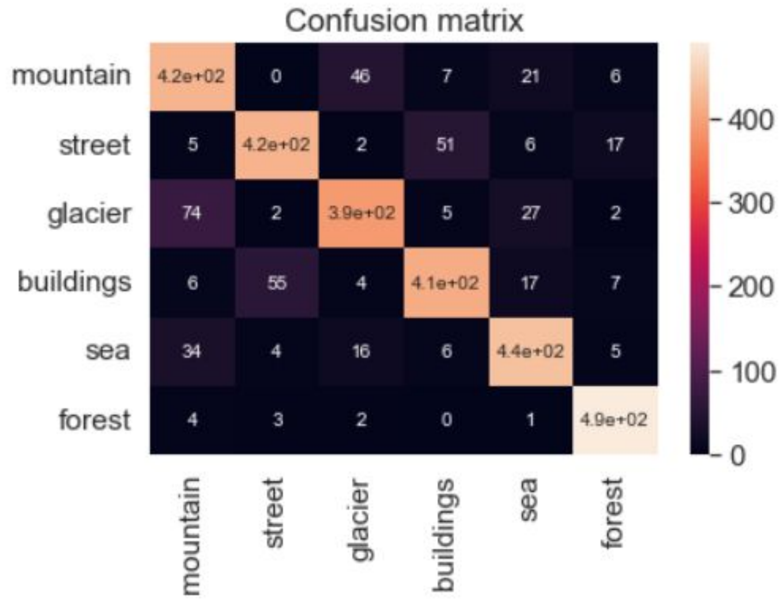
CNN Model 3 - Increasing no. of Convolution Block

```
model3 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu', input_shape = (150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.25),

    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.25),

    tf.keras.layers.Conv2D(64, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.25),

    tf.keras.layers.Conv2D(64, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.25),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(6, activation=tf.nn.softmax)
])
model3.summary()
```

CNN Model 3 - Increasing no. of Convolution Block

```
plot_accuracy_loss(history3)
```






```
test_loss = model3.evaluate(test_images, test_labels)
```

```
94/94 [==============================] - 7s 72ms/step - loss: 0.5443 - accuracy: 0.8550
```

CNN Model 4 - Tuning Learning Rate

```
from keras.callbacks import LearningRateScheduler
def step_decay_schedule(initial_lr=5e-4, decay_factor=0.95, step_size=2):
    '''
    Wrapper function to create a LearningRateScheduler with step decay schedule.
    '''

    def schedule(epoch):
        return initial_lr * (decay_factor ** np.floor(epoch / step_size))

    return LearningRateScheduler(schedule)
```

```
from tensorflow.keras.optimizers import SGD, Adam
optimizer = Adam(learning_rate=0.0005)
```

```
model4.compile(loss='sparse_categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
```

```
lr_sched = step_decay_schedule(initial_lr=5e-4, decay_factor=0.95, step_size=2)
```

```
history4 = model4.fit(train_images, train_labels, batch_size=128, epochs=64, validation_split = 0.2, callbacks=[lr_sched])
```

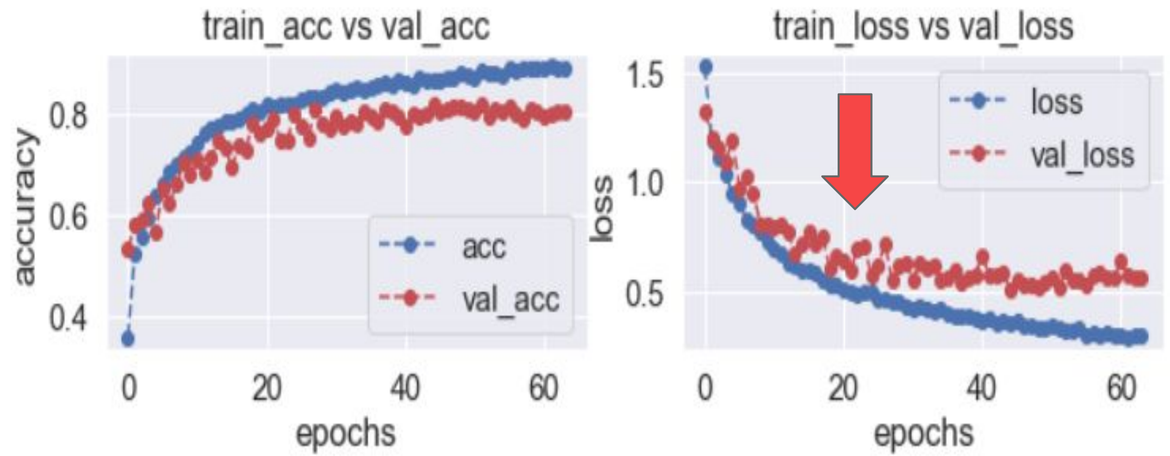
CNN Model 4 - Tuning Learning Rate

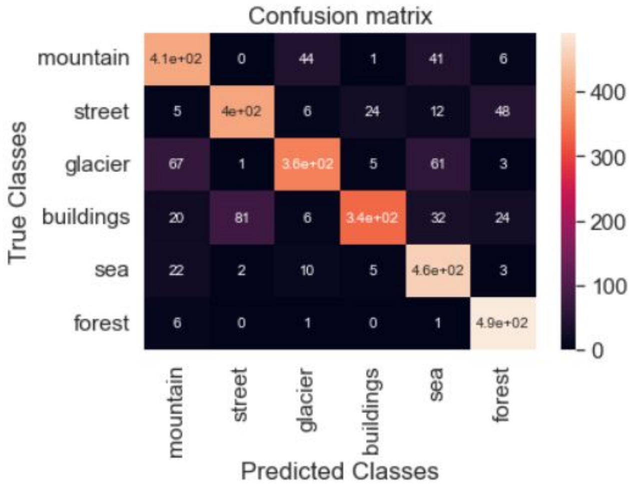
```
model4 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu', input_shape = (150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.25),

    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.25),

    tf.keras.layers.Conv2D(64, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.25),

    tf.keras.layers.Conv2D(64, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.25),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(6, activation=tf.nn.softmax)
])
model4.summary()
```

```
Epoch 1/64
75/75 [==============================] - 119s 2s/step - loss: 1.5251 - accuracy: 0.3590 - val_loss: 1.3193 - val_accuracy: 0.5329 - lr: 5.0000e-04
Epoch 2/64
75/75 [==============================] - 118s 2s/step - loss: 1.1912 - accuracy: 0.5249 - val_loss: 1.2050 - val_accuracy: 0.5817 - lr: 5.0000e-04
Epoch 3/64
75/75 [==============================] - 115s 2s/step - loss: 1.1072 - accuracy: 0.5579 - val_loss: 1.1549 - val_accuracy: 0.5925 - lr: 4.7500e-04
Epoch 4/64
75/75 [==============================] - 117s 2s/step - loss: 1.0311 - accuracy: 0.5969 - val_loss: 1.0951 - val_accuracy: 0.6229 - lr: 4.7500e-04
Epoch 5/64
75/75 [==============================] - 113s 2s/step - loss: 0.9465 - accuracy: 0.6367 - val_loss: 1.1897 - val_accuracy: 0.5667 - lr: 4.5125e-04
Epoch 6/64
75/75 [==============================] - 112s 1s/step - loss: 0.8984 - accuracy: 0.6608 - val_loss: 0.9742 - val_accuracy: 0.6525 - lr: 4.5125e-04
```

CNN Model 4 - Tuning Learning Rate

```
plot_accuracy_loss(history4)
```

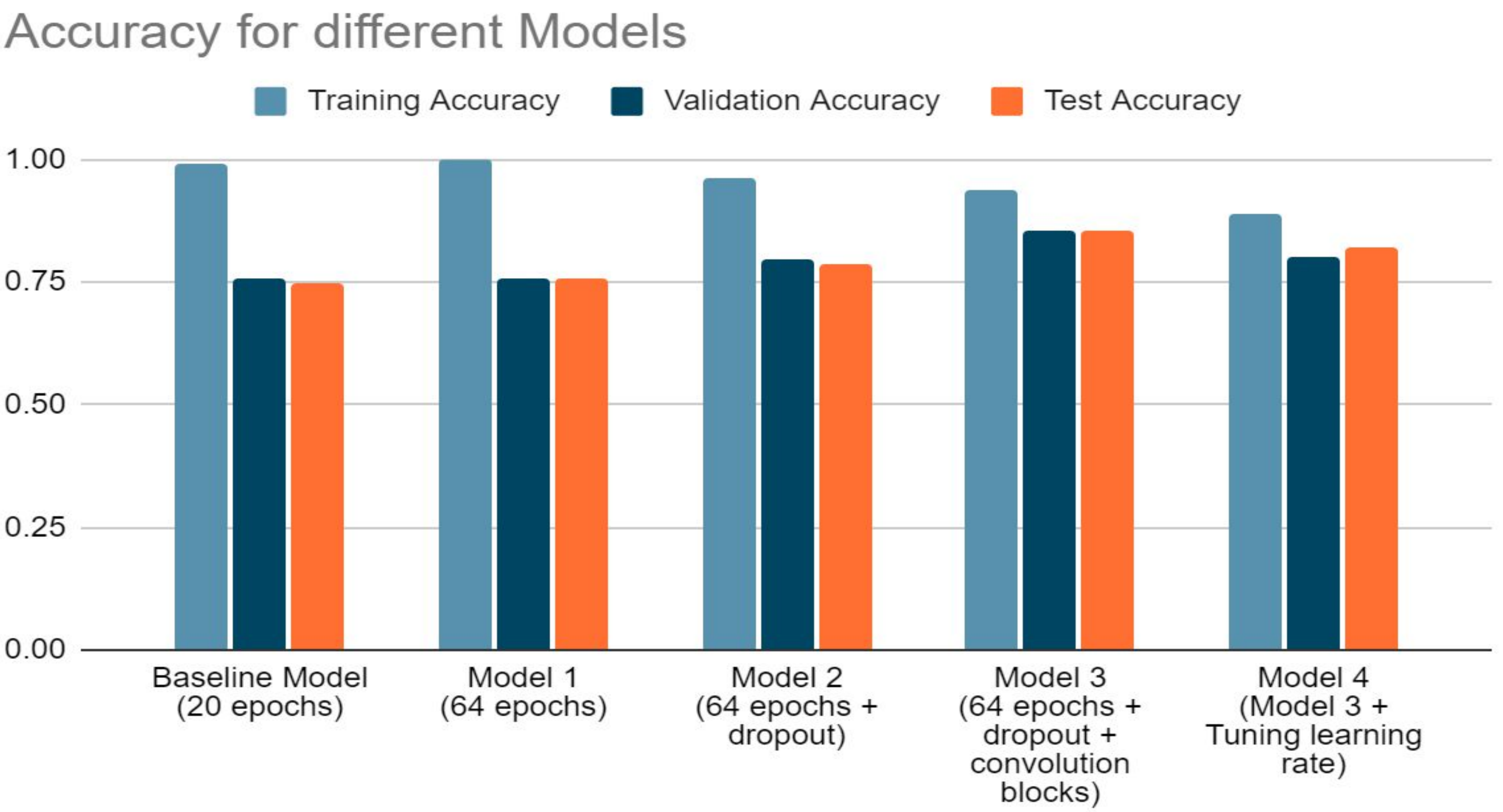




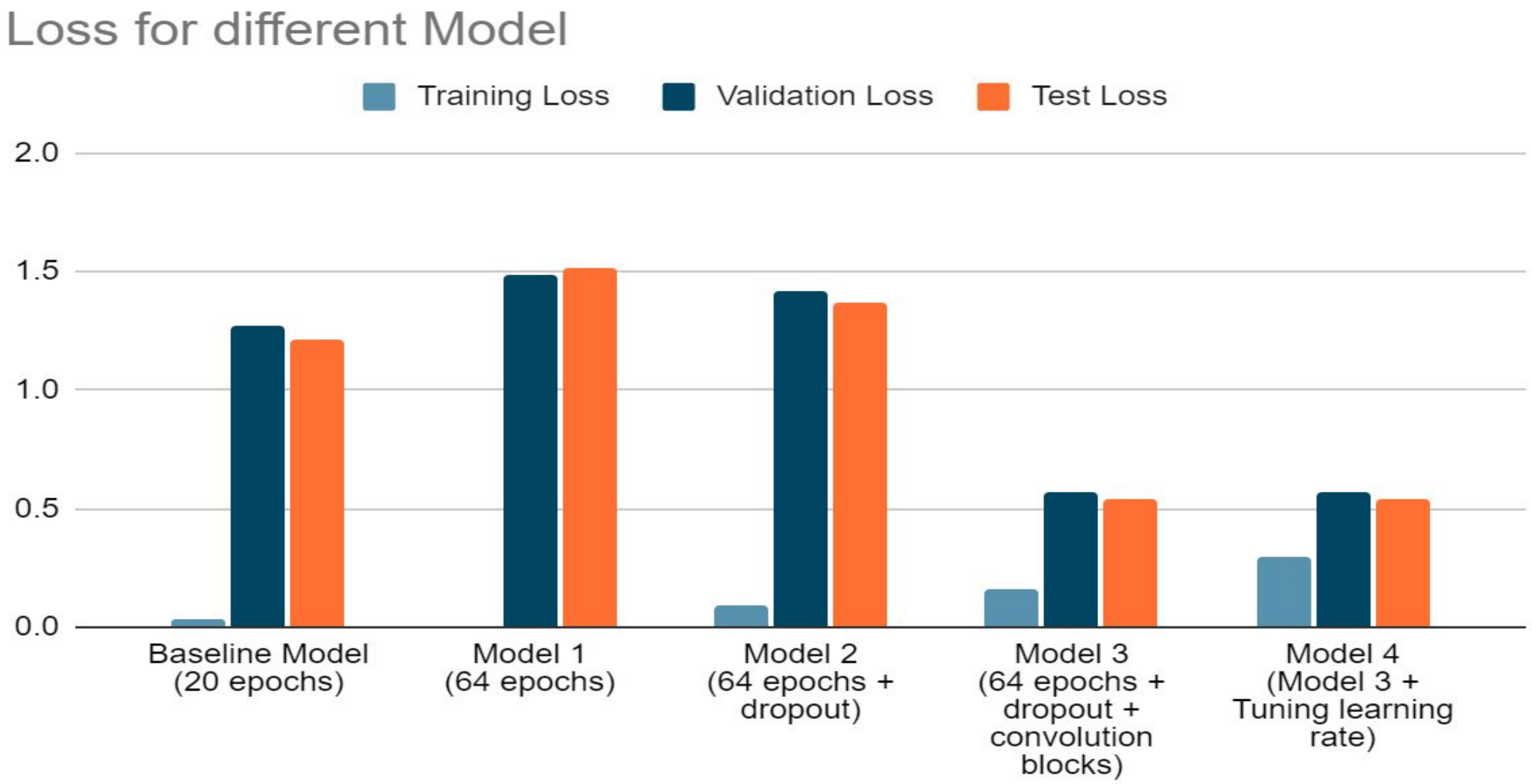

```
test_loss = model4.evaluate(test_images, test_labels)
```

```
94/94 [==============================] - 7s 71ms/step - loss: 0.5448 - accuracy: 0.8210
```

Conclusion


Accuracy for different Models
Training Accuracy
Validation Accuracy
Test Accuracy
1.00
0.75
0.50
0.25
0.00
Baseline Model (20 epochs)
Model 1 (64 epochs)
Model 2 (64 epochs + dropout)
Model 3 (64 epochs + dropout + convolution blocks)
Model 4 (Model 3 + Tuning learning rate)

Conclusion


Loss for different Model
Training Loss
Validation Loss
Test Loss
2.0
1.5
1.0
0.5
0.0
Baseline Model (20 epochs)
Model 1 (64 epochs)
Model 2 (64 epochs + dropout)
Model 3 (64 epochs + dropout + convolution blocks)
Model 4 (Model 3 + Tuning learning rate)

Takeaways

1. Multiple ways to increase accuracy and address issue of overfitting/losses - no. of epochs, dropout layer, convolution blocks, controlled learning rate

2. Increasing the number of epochs does not necessarily increase in training/validation accuracy of CNN model as it may lead to overfitting

3. Misclassification of images from the dataset set occurs, as they may include features from images of other types.

Thanks!

References

https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax

https://datascience.stackexchange.com/questions/109905/cannot-achieve-good-result-while-transfer-learning-cifar-10-on-resnet50-keras

https://stackoverflow.com/questions/39517431/should-we-do-learning-rate-decay-for-adam-optimizer

https://www.kaggle.com/code/vincee/intel-image-classification-cnn-keras

https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2