Data Description

Problem Statement :

- The aim of the project is to classify the road side signs category using CNN (with different architectures).

About Dataset:

- The dataset contains 80,000 synthetic images of road signs.
- Each image has a *.png format and a size of 224 x 224 pixels.
- There are 8 classes in the data that correspond to the categories of signs in Russia.
- For the test 3000 images per category.
- For train 7000 images per category.

In [2]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
import random
import os
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from tensorflow.keras.layers import BatchNormalization
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.models import Sequential, load_model
from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_array, array_to_img, load_img
from keras.layers import Conv2D, Dense, Flatten, Rescaling, AveragePooling2D, Dropout
from keras.layers import Dense, Activation, Dropout, Flatten, Conv2D, MaxPooling2D
from keras.layers import Dense, Conv2D, MaxPool2D , Flatten
```

In [3]:
```
# Import data path
data_dir = '/kaggle/input/russian-road-signs-categories-dataset'
train_path = '/kaggle/input/russian-road-signs-categories-dataset/train'
test_path = '/kaggle/input/russian-road-signs-categories-dataset/test'
```

In [4]:
```
#Initialize height and weight
height = 50
width = 50
```

In [5]:
```
# Initialize seeds and batch size
batch_size = 150
seed = 42
```

In [6]:
```
# Image rescaling
train_datagen = ImageDataGenerator(rescale=1./255,
                                   validation_split=0.2)
train_dataset = train_datagen.flow_from_directory(train_path,
                                                  target_size=(height, width),
                                                  batch_size=batch_size,
                                                  class_mode='categorical',
                                                  shuffle=True,
                                                  seed=seed,
                                                  color_mode='rgb',
                                                  interpolation='hamming',
                                                  subset='training')

test_datagen = ImageDataGenerator(rescale=1./255,
                                  validation_split=0.2)
test_dataset = test_datagen.flow_from_directory(train_path,
                                                target_size=(height, width),
                                                batch_size=batch_size,
                                                class_mode='categorical',
                                                shuffle=True,
                                                seed=seed,
                                                color_mode='rgb',
                                                interpolation='hamming',
                                                subset='validation')
```

```
Found 44800 images belonging to 8 classes.
Found 11200 images belonging to 8 classes.
```

We have 8 classes in training dataset in which 44800 images are there and 8 classes in testing dataset in which 11200 images are there

Basic CNN Model

In [7]:
```
model = keras.models.Sequential([
    keras.layers.Conv2D(filters=16, kernel_size=(5,5), activation='relu', input_shape=(height,width,3)),
    keras.layers.Conv2D(filters=32, kernel_size=(5,5), activation='relu'),
    keras.layers.MaxPool2D(pool_size=(2, 2)),
    keras.layers.BatchNormalization(axis=-1),

    keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
    keras.layers.Conv2D(filters=64, kernel_size=(3,3), activation='relu'),
    keras.layers.MaxPool2D(pool_size=(2, 2)),
    keras.layers.BatchNormalization(axis=-1),
    keras.layers.Dropout(rate=0.25),
    keras.layers.Flatten(),
    keras.layers.Dense(512, activation='relu'),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(rate=0.25),

    keras.layers.Dense(8, activation='softmax')
])
```

```
2022-12-27 13:03:40.660722: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS ha
d negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2022-12-27 13:03:40.741883: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS ha
d negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2022-12-27 13:03:40.742732: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS ha
d negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2022-12-27 13:03:40.743870: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAP
I Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations:  AVX2 AVX512F
FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2022-12-27 13:03:40.744157: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS ha
d negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2022-12-27 13:03:40.744859: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS ha
d negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2022-12-27 13:03:40.745523: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS ha
d negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2022-12-27 13:03:42.556841: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS ha
d negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2022-12-27 13:03:42.557787: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS ha
d negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2022-12-27 13:03:42.558520: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:937] successful NUMA node read from SysFS ha
d negative value (-1), but there must be at least one NUMA node, so returning NUMA node zero
2022-12-27 13:03:42.559100: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1510] Created device /job:localhost/replica:0/ta
sk:0/device:GPU:0 with 15401 MB memory:  -> device: 0, name: Tesla P100-PCIE-16GB, pci bus id: 0000:00:04.0, compute capabilit
y: 6.0
```

In [8]:
```
alpha=0.01
epochs=15
optim = keras.optimizers.Adam(learning_rate=0.01)
model.compile(optimizer = optim, loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

In [9]:
```
cnn = model.fit(train_dataset,
                steps_per_epoch = len(train_dataset),
                epochs = epochs,
                validation_data = test_dataset,
                validation_steps = len(test_dataset))
```

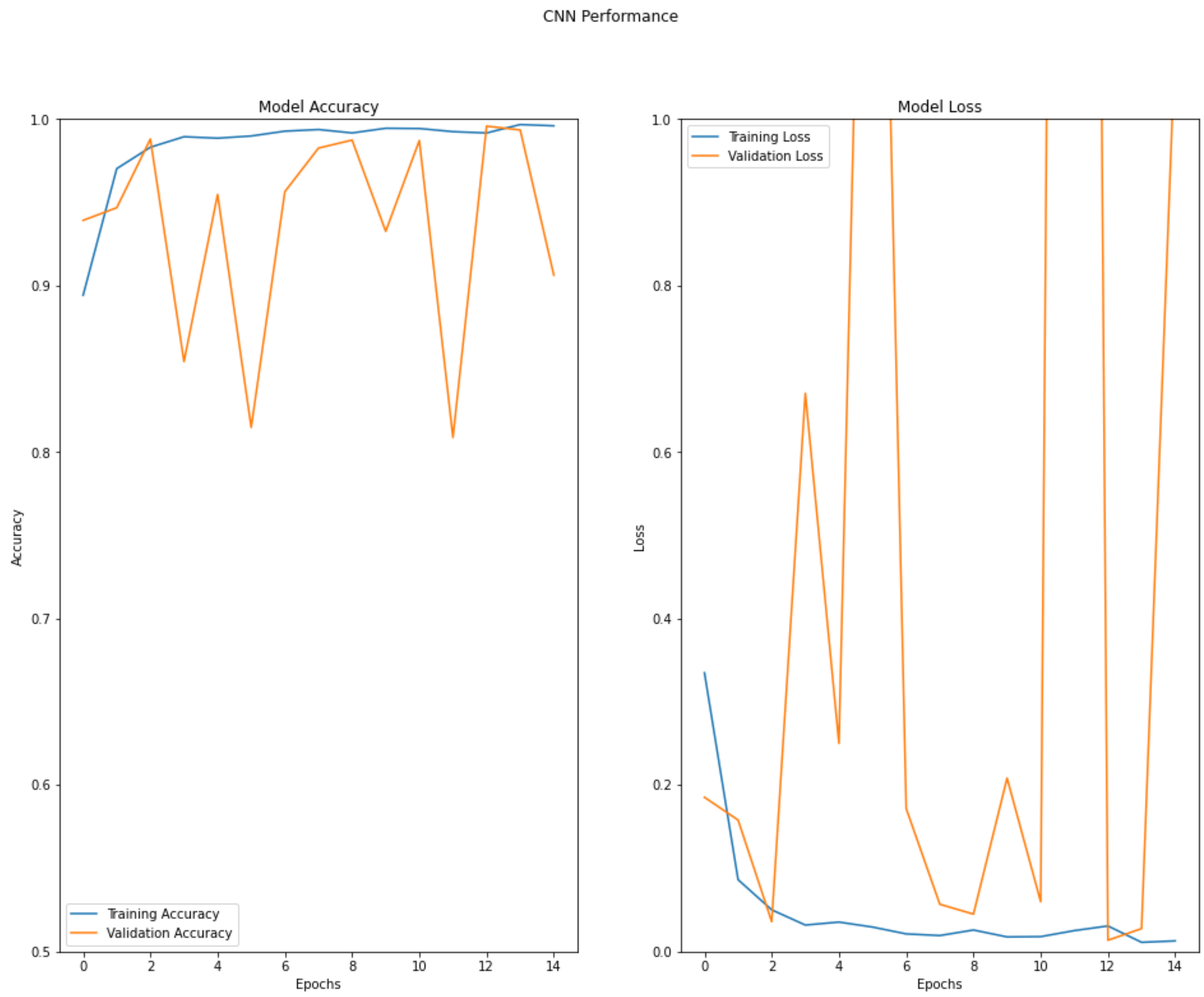
```
2022-12-27 13:03:43.967308: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the MLIR Optimization Passe
s are enabled (registered 2)

Epoch 1/15

2022-12-27 13:03:46.319641: I tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cuDNN version 8005

299/299 [==============================] - 306s 999ms/step - loss: 0.3349 - accuracy: 0.8943 - val_loss: 0.1854 - val_accuracy:
0.9393
Epoch 2/15
299/299 [==============================] - 172s 575ms/step - loss: 0.0864 - accuracy: 0.9703 - val_loss: 0.1581 - val_accuracy:
0.9469
Epoch 3/15
299/299 [==============================] - 175s 584ms/step - loss: 0.0502 - accuracy: 0.9832 - val_loss: 0.0359 - val_accuracy:
0.9881
Epoch 4/15
299/299 [==============================] - 171s 572ms/step - loss: 0.0319 - accuracy: 0.9895 - val_loss: 0.6708 - val_accuracy:
0.8545
Epoch 5/15
299/299 [==============================] - 170s 568ms/step - loss: 0.0355 - accuracy: 0.9886 - val_loss: 0.2501 - val_accuracy:
0.9548
Epoch 6/15
299/299 [==============================] - 170s 569ms/step - loss: 0.0295 - accuracy: 0.9900 - val_loss: 1.9660 - val_accuracy:
0.8149
Epoch 7/15
299/299 [==============================] - 169s 566ms/step - loss: 0.0213 - accuracy: 0.9929 - val_loss: 0.1718 - val_accuracy:
0.9565
Epoch 8/15
299/299 [==============================] - 169s 566ms/step - loss: 0.0193 - accuracy: 0.9938 - val_loss: 0.0569 - val_accuracy:
0.9827
Epoch 9/15
299/299 [==============================] - 170s 567ms/step - loss: 0.0259 - accuracy: 0.9918 - val_loss: 0.0450 - val_accuracy:
0.9875
Epoch 10/15
299/299 [==============================] - 169s 565ms/step - loss: 0.0176 - accuracy: 0.9946 - val_loss: 0.2084 - val_accuracy:
0.9327
Epoch 11/15
299/299 [==============================] - 169s 565ms/step - loss: 0.0180 - accuracy: 0.9944 - val_loss: 0.0599 - val_accuracy:
0.9872
Epoch 12/15
299/299 [==============================] - 169s 567ms/step - loss: 0.0252 - accuracy: 0.9926 - val_loss: 5.4399 - val_accuracy:
0.8088
Epoch 13/15
299/299 [==============================] - 174s 581ms/step - loss: 0.0308 - accuracy: 0.9918 - val_loss: 0.0137 - val_accuracy:
0.9959
Epoch 14/15
299/299 [==============================] - 170s 570ms/step - loss: 0.0111 - accuracy: 0.9968 - val_loss: 0.0276 - val_accuracy:
0.9936
Epoch 15/15
299/299 [==============================] - 173s 579ms/step - loss: 0.0129 - accuracy: 0.9961 - val_loss: 1.0953 - val_accuracy:
0.9063
```

In [10]:
```
# Graphical representation of Model accuracy and Loss
fig,ax=plt.subplots(1,2)
fig.set_size_inches(16,12)
performance = pd.DataFrame(cnn.history)
plt.figure(figsize=(10,7))
ax[1].plot(performance[['loss','val_loss']])
ax[1].legend(['Training Loss', 'Validation Loss'])
ax[1].set_title('Model Loss')
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Loss')
ax[1].set_ylim(0,1)
ax[0].plot(performance[['accuracy','val_accuracy']])
ax[0].legend(['Training Accuracy', 'Validation Accuracy'])
ax[0].set_title('Model Accuracy')
ax[0].set_xlabel('Epochs')
ax[0].set_ylabel('Accuracy')
ax[0].set_ylim(0.5,1)
fig.suptitle('CNN Performance')
plt.show()
```




```
<Figure size 720x504 with 0 Axes>
```

LeNet_5 Model

In [12]:
```
LeNet_5 = Sequential()
LeNet_5.add(Conv2D(filters=16, kernel_size=(5,5),  input_shape=(height,width,3))),
LeNet_5.add(BatchNormalization())
LeNet_5.add(Activation('relu'))
LeNet_5.add(AveragePooling2D())
LeNet_5.add(Conv2D(filters=32, kernel_size=(5,5)))
LeNet_5.add(BatchNormalization())
LeNet_5.add(Activation('relu'))
LeNet_5.add(AveragePooling2D())
LeNet_5.add(Flatten())
LeNet_5.add(Dense(units = 512))
LeNet_5.add(BatchNormalization())
LeNet_5.add(Activation('relu'))
LeNet_5.add(Dense(units = 120))
LeNet_5.add(BatchNormalization())
LeNet_5.add(Activation('relu'))
LeNet_5.add(Dense(units = 8))
LeNet_5.add(BatchNormalization())
LeNet_5.add(Activation('softmax'))
```

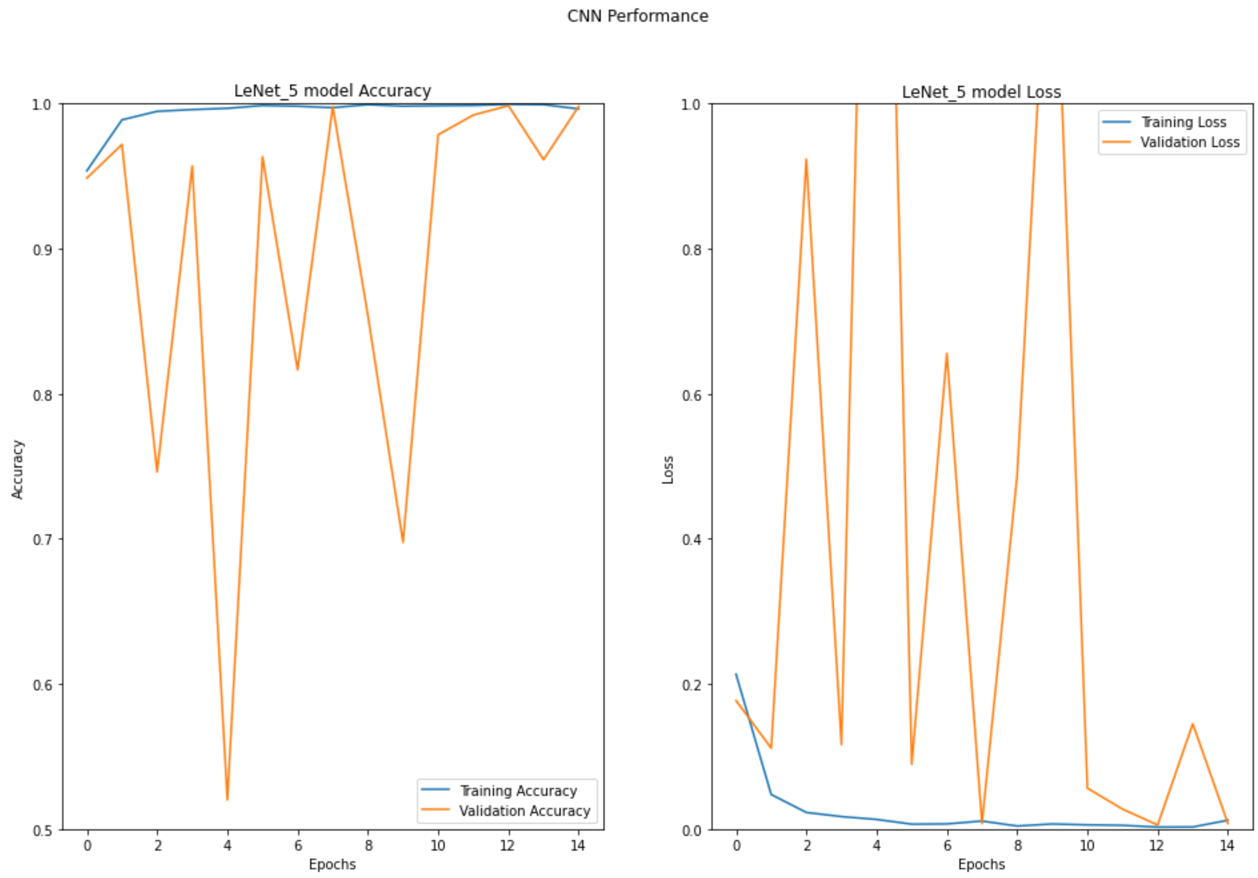
https://analyticsindiamag.com/complete-tutorial-on-lenet-5-guide-to-begin-with-cnns/ (https://analyticsindiamag.com/complete-tutorial-on-lenet-5-guide-to-begin-with-cnns/)

In [13]:
```
alpha=0.01
epochs=15
optim = keras.optimizers.Adam(learning_rate=0.01)
LeNet_5.compile(optimizer = optim, loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

In [14]:
```
cnn2 = LeNet_5.fit(train_dataset,
                steps_per_epoch = len(train_dataset),
                epochs = epochs,
                validation_data = test_dataset,
                validation_steps = len(test_dataset))
```

```
Epoch 1/15
299/299 [==============================] - 170s 565ms/step - loss: 0.2132 - accuracy: 0.9535 - val_loss: 0.1769 - val_accuracy:
0.9485
Epoch 2/15
299/299 [==============================] - 168s 562ms/step - loss: 0.0475 - accuracy: 0.9887 - val_loss: 0.1115 - val_accuracy:
0.9717
Epoch 3/15
299/299 [==============================] - 171s 571ms/step - loss: 0.0228 - accuracy: 0.9945 - val_loss: 0.9231 - val_accuracy:
0.7461
Epoch 4/15
299/299 [==============================] - 169s 567ms/step - loss: 0.0171 - accuracy: 0.9957 - val_loss: 0.1163 - val_accuracy:
0.9569
Epoch 5/15
299/299 [==============================] - 172s 577ms/step - loss: 0.0132 - accuracy: 0.9966 - val_loss: 2.1497 - val_accuracy:
0.5200
Epoch 6/15
299/299 [==============================] - 173s 577ms/step - loss: 0.0067 - accuracy: 0.9985 - val_loss: 0.0894 - val_accuracy:
0.9633
Epoch 7/15
299/299 [==============================] - 171s 572ms/step - loss: 0.0070 - accuracy: 0.9981 - val_loss: 0.6553 - val_accuracy:
0.8163
Epoch 8/15
299/299 [==============================] - 200s 670ms/step - loss: 0.0110 - accuracy: 0.9971 - val_loss: 0.0076 - val_accuracy:
0.9979
Epoch 9/15
299/299 [==============================] - 203s 680ms/step - loss: 0.0041 - accuracy: 0.9991 - val_loss: 0.4872 - val_accuracy:
0.8538
Epoch 10/15
299/299 [==============================] - 171s 573ms/step - loss: 0.0069 - accuracy: 0.9981 - val_loss: 1.3677 - val_accuracy:
0.6975
Epoch 11/15
299/299 [==============================] - 172s 575ms/step - loss: 0.0057 - accuracy: 0.9984 - val_loss: 0.0562 - val_accuracy:
0.9784
Epoch 12/15
299/299 [==============================] - 170s 570ms/step - loss: 0.0050 - accuracy: 0.9986 - val_loss: 0.0274 - val_accuracy:
0.9920
Epoch 13/15
299/299 [==============================] - 170s 570ms/step - loss: 0.0026 - accuracy: 0.9994 - val_loss: 0.0055 - val_accuracy:
0.9984
Epoch 14/15
299/299 [==============================] - 170s 568ms/step - loss: 0.0028 - accuracy: 0.9992 - val_loss: 0.1450 - val_accuracy:
0.9613
Epoch 15/15
299/299 [==============================] - 172s 576ms/step - loss: 0.0121 - accuracy: 0.9962 - val_loss: 0.0076 - val_accuracy:
0.9979
```

In [16]:
```
fig,ax=plt.subplots(1,2)
fig.set_size_inches(16,10)
performance = pd.DataFrame(cnn2.history)
plt.figure(figsize=(10,7))
ax[1].plot(performance[['loss','val_loss']])
ax[1].legend(['Training Loss', 'Validation Loss'])
ax[1].set_title('LeNet_5 model Loss')
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Loss')
ax[1].set_ylim(0,1)
ax[0].plot(performance[['accuracy','val_accuracy']])
ax[0].legend(['Training Accuracy', 'Validation Accuracy'])
ax[0].set_title('LeNet_5 model Accuracy')
ax[0].set_xlabel('Epochs')
ax[0].set_ylabel('Accuracy')
ax[0].set_ylim(0.5,1)
fig.suptitle('CNN Performance')
plt.show()
```




```
<Figure size 720x504 with 0 Axes>
```

AlexNet Model

In [17]:
```
AlexNet = Sequential()
AlexNet.add(Conv2D(filters=16, input_shape=(50,50,3), kernel_size=(11,11), strides=(4,4), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))
AlexNet.add(Conv2D(filters=32, kernel_size=(5, 5), strides=(1,1), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))
AlexNet.add(Conv2D(filters=48, kernel_size=(3,3), strides=(1,1), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(Conv2D(filters=64, kernel_size=(3,3), strides=(1,1), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(Conv2D(filters=64, kernel_size=(3,3), strides=(1,1), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

AlexNet.add(Flatten())

AlexNet.add(Dense(4096, input_shape=(32,32,3,)))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

AlexNet.add(Dropout(0.4))


AlexNet.add(Dense(4096))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

AlexNet.add(Dropout(0.4))


AlexNet.add(Dense(1000))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

AlexNet.add(Dropout(0.4))


AlexNet.add(Dense(8))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('softmax'))
```

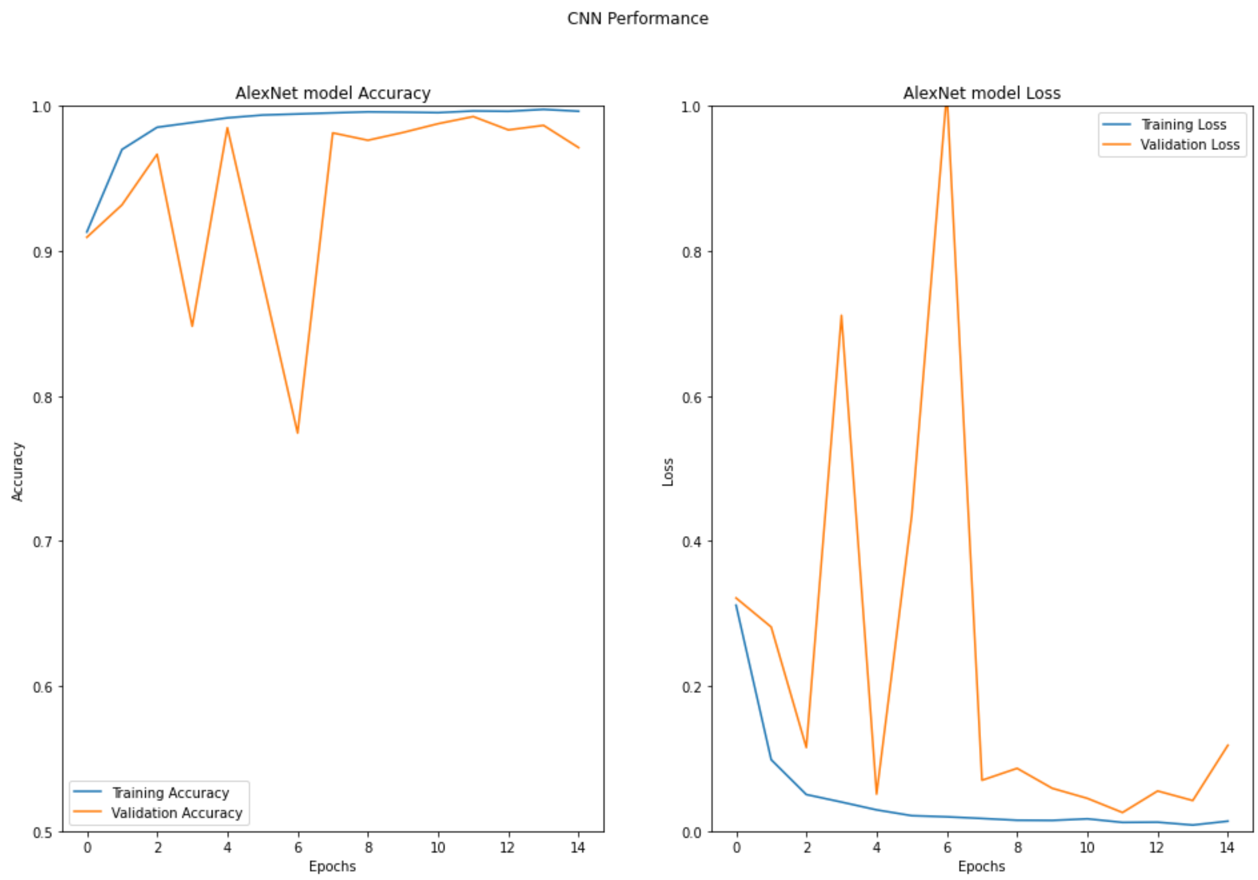
https://www.mygreatlearning.com/blog/alexnet-the-first-cnn-to-win-image-net/ (https://www.mygreatlearning.com/blog/alexnet-the-first-cnn-to-win-image-net/)

In [18]:
```
alpha=0.01
epochs=15
optim = keras.optimizers.Adam(learning_rate=0.01)
AlexNet.compile(optimizer = optim, loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

In [19]:
```
cnn3 = AlexNet.fit(train_dataset,
                steps_per_epoch = len(train_dataset),
                epochs = epochs,
                validation_data = test_dataset,
                validation_steps = len(test_dataset))
```

```
Epoch 1/15
299/299 [==============================] - 176s 583ms/step - loss: 0.3114 - accuracy: 0.9130 - val_loss: 0.3213 - val_accuracy: 
0.9094
Epoch 2/15
299/299 [==============================] - 172s 577ms/step - loss: 0.0984 - accuracy: 0.9699 - val_loss: 0.2813 - val_accuracy: 
0.9317
Epoch 3/15
299/299 [==============================] - 171s 573ms/step - loss: 0.0503 - accuracy: 0.9852 - val_loss: 0.1151 - val_accuracy: 
0.9666
Epoch 4/15
299/299 [==============================] - 171s 574ms/step - loss: 0.0400 - accuracy: 0.9885 - val_loss: 0.7111 - val_accuracy: 
0.8480
Epoch 5/15
299/299 [==============================] - 170s 571ms/step - loss: 0.0292 - accuracy: 0.9917 - val_loss: 0.0509 - val_accuracy: 
0.9849
Epoch 6/15
299/299 [==============================] - 170s 570ms/step - loss: 0.0212 - accuracy: 0.9936 - val_loss: 0.4350 - val_accuracy: 
0.8799
Epoch 7/15
299/299 [==============================] - 169s 567ms/step - loss: 0.0196 - accuracy: 0.9944 - val_loss: 1.0187 - val_accuracy: 
0.7744
Epoch 8/15
299/299 [==============================] - 170s 571ms/step - loss: 0.0173 - accuracy: 0.9951 - val_loss: 0.0700 - val_accuracy: 
0.9813
Epoch 9/15
299/299 [==============================] - 170s 568ms/step - loss: 0.0149 - accuracy: 0.9958 - val_loss: 0.0865 - val_accuracy: 
0.9762
Epoch 10/15
299/299 [==============================] - 170s 569ms/step - loss: 0.0146 - accuracy: 0.9956 - val_loss: 0.0589 - val_accuracy: 
0.9816
Epoch 11/15
299/299 [==============================] - 178s 596ms/step - loss: 0.0168 - accuracy: 0.9953 - val_loss: 0.0450 - val_accuracy: 
0.9877
Epoch 12/15
299/299 [==============================] - 207s 693ms/step - loss: 0.0119 - accuracy: 0.9965 - val_loss: 0.0255 - val_accuracy: 
0.9926
Epoch 13/15
299/299 [==============================] - 186s 623ms/step - loss: 0.0122 - accuracy: 0.9962 - val_loss: 0.0552 - val_accuracy: 
0.9834
Epoch 14/15
299/299 [==============================] - 184s 616ms/step - loss: 0.0083 - accuracy: 0.9975 - val_loss: 0.0421 - val_accuracy: 
0.9865
Epoch 15/15
299/299 [==============================] - 168s 561ms/step - loss: 0.0136 - accuracy: 0.9962 - val_loss: 0.1183 - val_accuracy: 
0.9712
```

In [20]:
```
fig,ax=plt.subplots(1,2)
fig.set_size_inches(16,10)
performance = pd.DataFrame(cnn3.history)
plt.figure(figsize=(10,7))
ax[1].plot(performance[['loss','val_loss']])
ax[1].legend(['Training Loss', 'Validation Loss'])
ax[1].set_title('AlexNet model Loss')
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Loss')
ax[1].set_ylim(0,1)
ax[0].plot(performance[['accuracy','val_accuracy']])
ax[0].legend(['Training Accuracy', 'Validation Accuracy'])
ax[0].set_title('AlexNet model Accuracy')
ax[0].set_xlabel('Epochs')
ax[0].set_ylabel('Accuracy')
ax[0].set_ylim(0.5,1)
fig.suptitle('CNN Performance')
plt.show()
```




```
<Figure size 720x504 with 0 Axes>
```

VGG Model

In [21]:
```
vgg = Sequential()
vgg.add(Conv2D(input_shape=(50,50,3),filters=16,kernel_size=(5,5),padding="same"))
vgg.add(Conv2D(filters=32,kernel_size=(5,5),padding="same", activation="relu"))
vgg.add(BatchNormalization())
vgg.add(Activation('relu'))
vgg.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
vgg.add(Conv2D(filters=32, kernel_size=(5,5), padding="same"))
vgg.add(BatchNormalization())
vgg.add(Activation('relu'))
vgg.add(Conv2D(filters=32, kernel_size=(5,5), padding="same"))
vgg.add(BatchNormalization())
vgg.add(Activation('relu'))
vgg.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
vgg.add(Conv2D(filters=48, kernel_size=(5,5), padding="same"))
vgg.add(BatchNormalization())
vgg.add(Activation('relu'))
vgg.add(Conv2D(filters=48, kernel_size=(5,5), padding="same", activation="relu"))
vgg.add(Conv2D(filters=48, kernel_size=(3,3), padding="same", activation="relu"))
vgg.add(BatchNormalization())
vgg.add(Activation('relu'))
vgg.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
vgg.add(Conv2D(filters=64, kernel_size=(5,5), padding="same"))
vgg.add(Conv2D(filters=64, kernel_size=(5,5), padding="same"))
vgg.add(BatchNormalization())
vgg.add(Activation('relu'))
vgg.add(Conv2D(filters=64, kernel_size=(5,5), padding="same"))
vgg.add(BatchNormalization())
vgg.add(Activation('relu'))
vgg.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
vgg.add(Conv2D(filters=80, kernel_size=(5,5), padding="same", activation="relu"))
vgg.add(Conv2D(filters=80, kernel_size=(5,5), padding="same", activation="relu"))
vgg.add(BatchNormalization())
vgg.add(Activation('relu'))
vgg.add(Conv2D(filters=80, kernel_size=(5,5), padding="same"))
vgg.add(BatchNormalization())
vgg.add(Activation('relu'))
vgg.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
vgg.add(Flatten())
vgg.add(Dense(units=4096))
vgg.add(BatchNormalization())
vgg.add(Activation('relu'))
vgg.add(Dense(units=1700,activation="relu"))
vgg.add(BatchNormalization())
vgg.add(Activation('relu'))
vgg.add(Dense(units=8))
vgg.add(BatchNormalization())
vgg.add(Activation('softmax'))
```

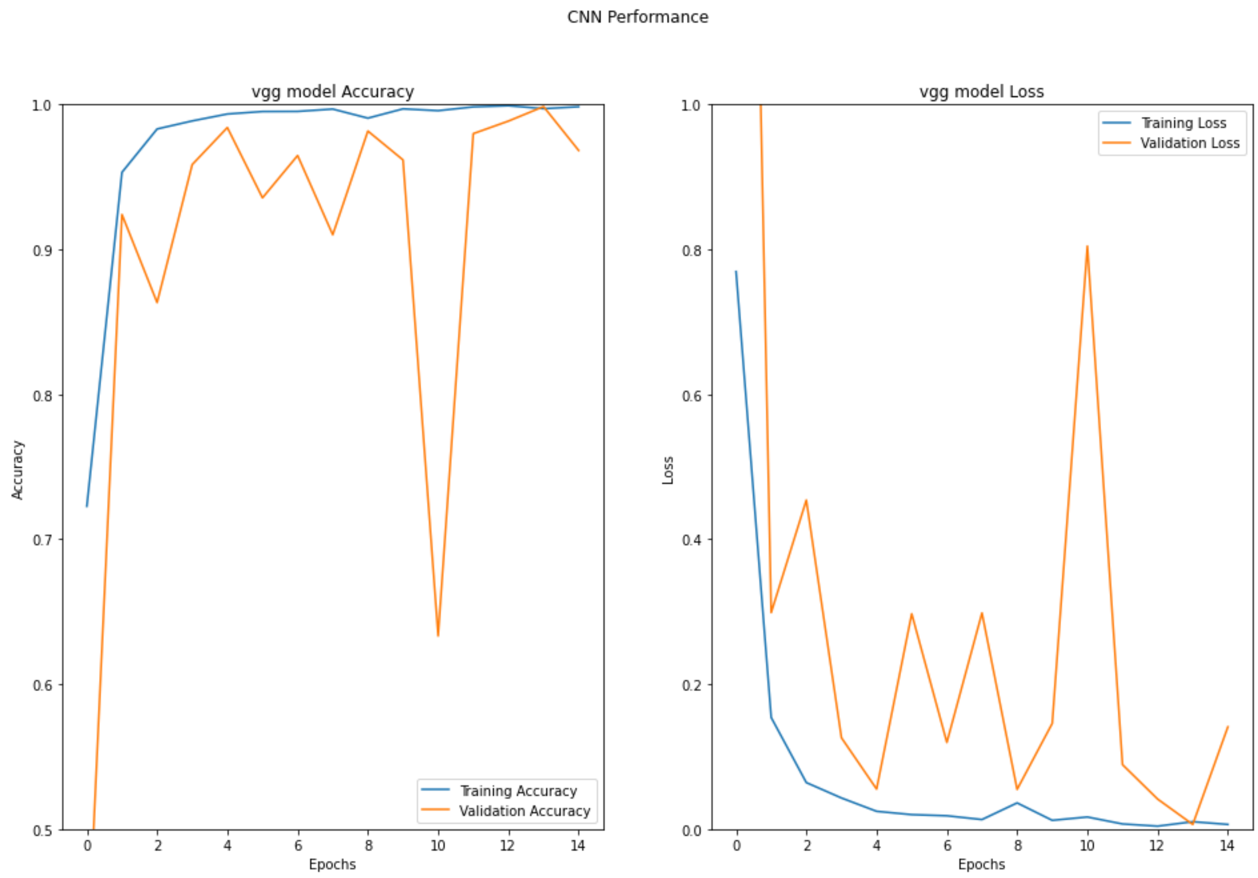
[https://www.geeksforgeeks.org/vgg-16-cnn-model/ (https://www.geeksforgeeks.org/vgg-16-cnn-model/)](https://www.geeksforgeeks.org/vgg-16-cnn-model/)

In [22]:
```
alpha=0.01
epochs=15
optim = keras.optimizers.Adam(learning_rate=0.01)
vgg.compile(optimizer = optim, loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

In [23]:
```
cnn4 = vgg.fit(train_dataset,
                steps_per_epoch = len(train_dataset),
                epochs = epochs,
                validation_data = test_dataset,
                validation_steps = len(test_dataset))
```

```
Epoch 1/15
299/299 [==============================] - 172s 571ms/step - loss: 0.7692 - accuracy: 0.7227 - val_loss: 2.6579 - val_accuracy:
0.3986
Epoch 2/15
299/299 [==============================] - 172s 575ms/step - loss: 0.1539 - accuracy: 0.9530 - val_loss: 0.2986 - val_accuracy:
0.9239
Epoch 3/15
299/299 [==============================] - 173s 579ms/step - loss: 0.0641 - accuracy: 0.9829 - val_loss: 0.4539 - val_accuracy:
0.8632
Epoch 4/15
299/299 [==============================] - 177s 592ms/step - loss: 0.0429 - accuracy: 0.9885 - val_loss: 0.1260 - val_accuracy:
0.9584
Epoch 5/15
299/299 [==============================] - 172s 575ms/step - loss: 0.0245 - accuracy: 0.9933 - val_loss: 0.0552 - val_accuracy:
0.9839
Epoch 6/15
299/299 [==============================] - 170s 569ms/step - loss: 0.0199 - accuracy: 0.9950 - val_loss: 0.2970 - val_accuracy:
0.9354
Epoch 7/15
299/299 [==============================] - 169s 564ms/step - loss: 0.0183 - accuracy: 0.9951 - val_loss: 0.1196 - val_accuracy:
0.9646
Epoch 8/15
299/299 [==============================] - 170s 568ms/step - loss: 0.0131 - accuracy: 0.9967 - val_loss: 0.2982 - val_accuracy:
0.9100
Epoch 9/15
299/299 [==============================] - 171s 571ms/step - loss: 0.0361 - accuracy: 0.9904 - val_loss: 0.0547 - val_accuracy:
0.9815
Epoch 10/15
299/299 [==============================] - 169s 566ms/step - loss: 0.0121 - accuracy: 0.9968 - val_loss: 0.1458 - val_accuracy:
0.9616
Epoch 11/15
299/299 [==============================] - 169s 566ms/step - loss: 0.0166 - accuracy: 0.9956 - val_loss: 0.8044 - val_accuracy:
0.6332
Epoch 12/15
299/299 [==============================] - 169s 566ms/step - loss: 0.0070 - accuracy: 0.9982 - val_loss: 0.0889 - val_accuracy:
0.9797
Epoch 13/15
299/299 [==============================] - 169s 566ms/step - loss: 0.0040 - accuracy: 0.9990 - val_loss: 0.0414 - val_accuracy:
0.9884
Epoch 14/15
299/299 [==============================] - 168s 563ms/step - loss: 0.0101 - accuracy: 0.9970 - val_loss: 0.0064 - val_accuracy:
0.9984
Epoch 15/15
299/299 [==============================] - 170s 568ms/step - loss: 0.0064 - accuracy: 0.9983 - val_loss: 0.1410 - val_accuracy:
0.9681
```

In [24]:
```python
# Graphical representation of acuuracy and Loss
fig,ax=plt.subplots(1,2)
fig.set_size_inches(16,10)
performance = pd.DataFrame(cnn4.history)
plt.figure(figsize=(10,7))
ax[1].plot(performance[['loss','val_loss']])
ax[1].legend(['Training Loss', 'Validation Loss'])
ax[1].set_title('vgg model Loss')
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Loss')
ax[1].set_ylim(0,1)
ax[0].plot(performance[['accuracy','val_accuracy']])
ax[0].legend(['Training Accuracy', 'Validation Accuracy'])
ax[0].set_title('vgg model Accuracy')
ax[0].set_xlabel('Epochs')
ax[0].set_ylabel('Accuracy')
ax[0].set_ylim(0.5,1)
fig.suptitle('CNN Performance')
plt.show()
```




```
<Figure size 720x504 with 0 Axes>
```

In [25]:
```python
# prediction on single image
from keras.preprocessing import image

val_image = image.load_img('/kaggle/input/russian-road-signs-categories-dataset/test/3/3_7011.png',target_size=(height, width))
val_image = image.img_to_array(val_image)
val_image = np.expand_dims(val_image,axis=0)
```

In [26]:
```python
result = model.predict(val_image)
result
```

Out[26]:
```
array([[0., 0., 1., 0., 0., 0., 0., 0.]], dtype=float32)
```

In [27]:
```python
train_dataset.class_indices
```

Out[27]:
```
{'1': 0, '2': 1, '3': 2, '4': 3, '5': 4, '6': 5, '7': 6, '8': 7}
```

Conclusion:

- Here we use four types of cnn architecture, among all Lenet_5 cnn architecture gives better accuracy with minimun loss.

Note - This dataset has 8 directories for train and test and some images are present in every directory so, The prediction may be wrong sometimes.