

✓ DOGS VS CATS CNN Classification

✓ Data Loading from Kaggle using Kaggle Token

```
from google.colab import files
files.upload()
```



Choose Files

No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving kaggle.json to kaggle.json

```
{'kaggle.json': b'{"username": "namano", "key": "51ea70999686195f63d37021bd453ee5"}'}
```

```
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

Data Downloading from kaggle and then unzipping the Train and Test data

```
!kaggle competitions download -c dogs-vs-cats-redux-kernels-edition
```



Downloading dogs-vs-cats-redux-kernels-edition.zip to /content

100% 813M/814M [00:09<00:00, 103MB/s]

100% 814M/814M [00:09<00:00, 86.9MB/s]

```
!ls
```



dogs-vs-cats-redux-kernels-edition.zip kaggle.json sample_data

```
!unzip dogs-vs-cats-redux-kernels-edition.zip -d ./dogs-vs-cats
```

```
!unzip ./dogs-vs-cats/train.zip -d ./dogs-vs-cats/train
```

```
!unzip ./dogs-vs-cats/test.zip -d ./dogs-vs-cats/test
```



```
inflating: ./dogs-vs-cats/test/test/569.jpg
inflating: ./dogs-vs-cats/test/test/5690.jpg
inflating: ./dogs-vs-cats/test/test/5691.jpg
inflating: ./dogs-vs-cats/test/test/5692.jpg
inflating: ./dogs-vs-cats/test/test/5693.jpg
inflating: ./dogs-vs-cats/test/test/5694.jpg
inflating: ./dogs-vs-cats/test/test/5695.jpg
inflating: ./dogs-vs-cats/test/test/5696.jpg
inflating: ./dogs-vs-cats/test/test/5697.jpg
inflating: ./dogs-vs-cats/test/test/5698.jpg
inflating: ./dogs-vs-cats/test/test/5699.jpg
inflating: ./dogs-vs-cats/test/test/57.jpg
inflating: ./dogs-vs-cats/test/test/570.jpg
inflating: ./dogs-vs-cats/test/test/5700.jpg
inflating: ./dogs-vs-cats/test/test/5701.jpg
inflating: ./dogs-vs-cats/test/test/5702.jpg
inflating: ./dogs-vs-cats/test/test/5703.jpg
inflating: ./dogs-vs-cats/test/test/5704.jpg
inflating: ./dogs-vs-cats/test/test/5705.jpg
inflating: ./dogs-vs-cats/test/test/5706.jpg
inflating: ./dogs-vs-cats/test/test/5707.jpg
inflating: ./dogs-vs-cats/test/test/5708.jpg
inflating: ./dogs-vs-cats/test/test/5709.jpg
inflating: ./dogs-vs-cats/test/test/571.jpg
inflating: ./dogs-vs-cats/test/test/5710.jpg
inflating: ./dogs-vs-cats/test/test/5711.jpg
inflating: ./dogs-vs-cats/test/test/5712.jpg
inflating: ./dogs-vs-cats/test/test/5713.jpg
inflating: ./dogs-vs-cats/test/test/5714.jpg
inflating: ./dogs-vs-cats/test/test/5715.jpg
inflating: ./dogs-vs-cats/test/test/5716.jpg
inflating: ./dogs-vs-cats/test/test/5717.jpg
inflating: ./dogs-vs-cats/test/test/5718.jpg
inflating: ./dogs-vs-cats/test/test/5719.jpg
inflating: ./dogs-vs-cats/test/test/572.jpg
inflating: ./dogs-vs-cats/test/test/5720.jpg
inflating: ./dogs-vs-cats/test/test/5721.jpg
inflating: ./dogs-vs-cats/test/test/5722.jpg
inflating: ./dogs-vs-cats/test/test/5723.jpg
inflating: ./dogs-vs-cats/test/test/5724.jpg
inflating: ./dogs-vs-cats/test/test/5725.jpg
inflating: ./dogs-vs-cats/test/test/5726.jpg
inflating: ./dogs-vs-cats/test/test/5727.jpg
inflating: ./dogs-vs-cats/test/test/5728.jpg
inflating: ./dogs-vs-cats/test/test/5729.jpg
inflating: ./dogs-vs-cats/test/test/573.jpg
inflating: ./dogs-vs-cats/test/test/5730.jpg
inflating: ./dogs-vs-cats/test/test/5731.jpg
inflating: ./dogs-vs-cats/test/test/5732.jpg
```

✓ Setting up the Tensorflow and Keras Environment

```
#!/pip install tensorflow keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

✓ Task 1

Load and preprocess the dataset without validation data. We'll only use a training set and a test set.

```
# Set up data augmentation for training
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Rescale validation data (without augmentation)
test_datagen = ImageDataGenerator(rescale=1./255)

# Load training, validation, and test datasets
train_generator = train_datagen.flow_from_directory(
    'dogs-vs-cats/train',
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary',
    subset='training'
)
```

➡ Found 25000 images belonging to 1 classes.

```
test_generator = test_datagen.flow_from_directory(
    'dogs-vs-cats/test',
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary'
)
```

➡ Found 12500 images belonging to 1 classes.

Use the ImageDataGenerator from Keras to augment the training data. This helps prevent overfitting when training with small datasets. Load the dataset from directories, rescale the pixel values, and

perform random transformations (rotation, shift, flip) to improve model generalization. No validation data is used, so the model is trained directly on 1000 training samples and evaluated on 500 test samples.

```
from keras import layers, models
```

```
# Build CNN model
```

```
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    layers.MaxPooling2D(2, 2),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D(2, 2),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D(2, 2),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D(2, 2),
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])
```

```
# Compile the model
```

```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
↳ /usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107:
    super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Create a CNN model from scratch using layers like convolutional layers, max-pooling, and fully connected layers. The model is compiled using the Adam optimizer and binary cross-entropy loss since the task is binary classification (cats vs. dogs).

```
history = model.fit(
    train_generator,
    steps_per_epoch=30, # 1000 training samples / batch size of 20
    epochs=20
)
```

```
↳ Epoch 1/20
30/30 ————— 37s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 2/20
30/30 ————— 32s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 3/20
30/30 ————— 35s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00
```

```

Epoch 4/20
30/30 ————— 39s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 5/20
30/30 ————— 33s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 6/20
30/30 ————— 32s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 7/20
30/30 ————— 35s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 8/20
30/30 ————— 33s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 9/20
30/30 ————— 32s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 10/20
30/30 ————— 32s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 11/20
30/30 ————— 35s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 12/20
30/30 ————— 32s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 13/20
30/30 ————— 32s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 14/20
30/30 ————— 34s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 15/20
30/30 ————— 32s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 16/20
30/30 ————— 32s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 17/20
30/30 ————— 34s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 18/20
30/30 ————— 32s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 19/20
30/30 ————— 32s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 20/20
30/30 ————— 33s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00

```

```

test_loss1, test_acc1 = model.evaluate(test_generator, steps=25) # 500 test samples / batch
print(f'Test accuracy: {test_acc1}')

```

```

➡ 25/25 ————— 131s 5s/step - accuracy: 1.0000 - loss: 1.6071e-24
Test accuracy: 1.0

```

Train the model on the training dataset with the specified number of epochs (e.g., 30 epochs). Monitor the performance after each epoch by recording training accuracy and loss.

After training, evaluate the model's performance on the test set (500 images). Save the test accuracy and loss for later comparison.

✓ Task 2

Repeat the data augmentation and loading process, but this time increase the training sample size (e.g., to 2000 images). Again, rescale and augment the training data, and keep the test set the same (500 images).

```
train_generator = train_datagen.flow_from_directory(
    'dogs-vs-cats/train',
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary'
)

history = model.fit(
    train_generator,
    steps_per_epoch=40, # For 2000 samples, steps_per_epoch = 2000 / batch size
    epochs=20
)
```

Found 25000 images belonging to 1 classes.

Epoch 1/20	40/40	44s	1s/step	- accuracy: 1.0000	- loss: 0.0000e+00
Epoch 2/20	40/40	43s	1s/step	- accuracy: 1.0000	- loss: 0.0000e+00
Epoch 3/20	40/40	43s	1s/step	- accuracy: 1.0000	- loss: 0.0000e+00
Epoch 4/20	40/40	43s	1s/step	- accuracy: 1.0000	- loss: 0.0000e+00
Epoch 5/20	40/40	44s	1s/step	- accuracy: 1.0000	- loss: 0.0000e+00
Epoch 6/20	40/40	45s	1s/step	- accuracy: 1.0000	- loss: 0.0000e+00
Epoch 7/20	40/40	47s	1s/step	- accuracy: 1.0000	- loss: 0.0000e+00
Epoch 8/20	40/40	43s	1s/step	- accuracy: 1.0000	- loss: 0.0000e+00
Epoch 9/20	40/40	43s	1s/step	- accuracy: 1.0000	- loss: 0.0000e+00
Epoch 10/20	40/40	43s	1s/step	- accuracy: 1.0000	- loss: 0.0000e+00
Epoch 11/20	40/40	43s	1s/step	- accuracy: 1.0000	- loss: 0.0000e+00
Epoch 12/20	40/40	43s	1s/step	- accuracy: 1.0000	- loss: 0.0000e+00
Epoch 13/20	40/40	43s	1s/step	- accuracy: 1.0000	- loss: 0.0000e+00
Epoch 14/20	40/40	43s	1s/step	- accuracy: 1.0000	- loss: 0.0000e+00
Epoch 15/20	40/40	43s	1s/step	- accuracy: 1.0000	- loss: 0.0000e+00
Epoch 16/20					

```

40/40 ————— 43s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 17/20
40/40 ————— 43s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 18/20
40/40 ————— 43s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 19/20
40/40 ————— 44s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00
Epoch 20/20
40/40 ————— 43s 1s/step - accuracy: 1.0000 - loss: 0.0000e+00
25/25 ————— 6s 244ms/step - accuracy: 1.0000 - loss: 0.0000e+00
Test accuracy: 1.0

```

Train the CNN model from scratch, using the larger dataset. Monitor training accuracy and loss over the epochs, similar to Step 1, but with more data to improve generalization.

```

# Evaluate on test set
test_loss2, test_acc2 = model.evaluate(test_generator, steps=25)
print(f'Test accuracy: {test_acc2}')

```

```

➡ 25/25 ————— 126s 5s/step - accuracy: 1.0000 - loss: 4.3633e-24
Test accuracy: 1.0

```

Evaluate the performance of the model on the same 500 test samples. Save the test accuracy and loss to compare with previous results.

```

from keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
                    include_top=False,
                    input_shape=(150, 150, 3))

# Freeze the convolutional base
conv_base.trainable = False

```

```

➡ Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16
58889256/58889256 ————— 0s 0us/step

```



✓ Task 3

Load a pretrained model such as VGG16, which has been trained on a large dataset (ImageNet). Freeze the convolutional base to prevent the weights from being updated during training. Add fully

connected layers on top of the pretrained model to adapt it to the binary classification task (cats vs. dogs). Compile the model and set it up for training.

```
from keras import models, layers

model = models.Sequential([
    conv_base,
    layers.Flatten(),
    layers.Dense(256, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Train the pretrained model on the same training set (e.g., 1000 samples), and monitor performance. Evaluate the model on the test set after training and record the accuracy and loss. Unfreeze the last few layers of the pretrained model and fine-tune them for better performance. Train the model again on the same training data and evaluate it on the test set to see if the performance improves. Record the test accuracy and loss.

```
history = model.fit(
    train_generator,
    steps_per_epoch=20, # Use the same training data size as in Step 1
    epochs=20
)
```

```
➡ Epoch 1/20
20/20 ————— 100s 5s/step - accuracy: 0.8263 - loss: 0.1787
Epoch 2/20
20/20 ————— 96s 5s/step - accuracy: 1.0000 - loss: 4.6869e-25
Epoch 3/20
20/20 ————— 96s 5s/step - accuracy: 1.0000 - loss: 2.3599e-24
Epoch 4/20
20/20 ————— 95s 5s/step - accuracy: 1.0000 - loss: 7.8883e-25
Epoch 5/20
20/20 ————— 95s 5s/step - accuracy: 1.0000 - loss: 1.3369e-23
Epoch 6/20
20/20 ————— 99s 5s/step - accuracy: 1.0000 - loss: 7.8550e-24
Epoch 7/20
20/20 ————— 96s 5s/step - accuracy: 1.0000 - loss: 1.8658e-26
Epoch 8/20
20/20 ————— 98s 5s/step - accuracy: 1.0000 - loss: 5.5932e-27
```



```

Epoch 9/20
20/20 ————— 98s 5s/step - accuracy: 1.0000 - loss: 2.5665e-25
Epoch 10/20
20/20 ————— 96s 5s/step - accuracy: 1.0000 - loss: 1.3347e-21
Epoch 11/20
20/20 ————— 96s 5s/step - accuracy: 1.0000 - loss: 7.3386e-25
Epoch 12/20
20/20 ————— 98s 5s/step - accuracy: 1.0000 - loss: 4.4034e-25
Epoch 13/20
20/20 ————— 100s 5s/step - accuracy: 1.0000 - loss: 1.8396e-23
Epoch 14/20
20/20 ————— 98s 5s/step - accuracy: 1.0000 - loss: 2.2132e-25
Epoch 15/20
20/20 ————— 97s 5s/step - accuracy: 1.0000 - loss: 3.1738e-24
Epoch 16/20
20/20 ————— 97s 5s/step - accuracy: 1.0000 - loss: 3.0574e-26
Epoch 17/20
20/20 ————— 97s 5s/step - accuracy: 1.0000 - loss: 2.4801e-26
Epoch 18/20
20/20 ————— 96s 5s/step - accuracy: 1.0000 - loss: 7.2306e-26
Epoch 19/20
20/20 ————— 96s 5s/step - accuracy: 1.0000 - loss: 2.5608e-25
Epoch 20/20
20/20 ————— 96s 5s/step - accuracy: 1.0000 - loss: 4.8450e-25
25/25 ————— 120s 5s/step - accuracy: 1.0000 - loss: 1.2320e-25
Test accuracy: 1.0

```

Evaluate on test set

```

test_loss_pretrained, test_acc_pretrained = model.evaluate(test_generator, steps=25)
print(f'Test accuracy: {test_acc_pretrained}')

```

```

⇒ 25/25 ————— 122s 5s/step - accuracy: 1.0000 - loss: 5.5370e-23
Test accuracy: 1.0

```

```

conv_base.trainable = True
for layer in conv_base.layers[:-4]:
    layer.trainable = False

```

```

# Recompile and retrain the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

```

```

history = model.fit(
    train_generator,
    steps_per_epoch=30,
    epochs=10
)

```

```

→ Epoch 1/10
30/30 ————— 172s 6s/step - accuracy: 1.0000 - loss: 4.7572e-23
Epoch 2/10
30/30 ————— 169s 6s/step - accuracy: 1.0000 - loss: 2.9459e-25
Epoch 3/10
30/30 ————— 168s 6s/step - accuracy: 1.0000 - loss: 9.1313e-24
Epoch 4/10
30/30 ————— 169s 6s/step - accuracy: 1.0000 - loss: 8.5049e-25
Epoch 5/10
30/30 ————— 168s 6s/step - accuracy: 1.0000 - loss: 2.1705e-24
Epoch 6/10
30/30 ————— 169s 6s/step - accuracy: 1.0000 - loss: 6.6683e-26
Epoch 7/10
30/30 ————— 170s 6s/step - accuracy: 1.0000 - loss: 1.8261e-22
Epoch 8/10
30/30 ————— 174s 6s/step - accuracy: 1.0000 - loss: 1.9726e-23
Epoch 9/10
30/30 ————— 177s 6s/step - accuracy: 1.0000 - loss: 4.1327e-23
Epoch 10/10
30/30 ————— 174s 6s/step - accuracy: 1.0000 - loss: 5.7232e-25
25/25 ————— 121s 5s/step - accuracy: 1.0000 - loss: 1.7458e-24
Test accuracy after fine-tuning: 1.0

```

```
# Evaluate on test set
```

```
test_loss_finetuned, test_acc_finetuned = model.evaluate(test_generator, steps=25)
print(f'Test accuracy after fine-tuning: {test_acc_finetuned}')
```

```

→ 25/25 ————— 131s 5s/step - accuracy: 1.0000 - loss: 5.0978e-21
Test accuracy after fine-tuning: 1.0

```

✓ Task 4

✓ Table and Summary

```
import pandas as pd
```

```
# Creating a DataFrame to store results
```


```

results = pd.DataFrame({
    'Model': ['CNN from Scratch', 'CNN from Scratch', 'Pretrained CNN', 'Pretrained CNN Fine'],
    'Training Sample Size': [1000, 2000, 1000, 1000],
    'Test Accuracy': [test_acc1, test_acc2, test_acc_pretrained, test_acc_finetuned],
    'Test Loss': [test_loss1, test_loss2, test_loss_pretrained, test_loss_finetuned]
})

```

```
# Display the DataFrame as a table
```

```
print(results)
```



	Model	Training Sample Size	Test Accuracy \
0	CNN from Scratch	1000	1.0
1	CNN from Scratch	2000	1.0
2	Pretrained CNN	1000	1.0
3	Pretrained CNN Fine-tuned	1000	1.0

	Test Loss
0	2.169746e-24
1	1.807756e-24
2	1.384881e-22
3	3.204093e-20

CNN from scratch (1000 training samples) CNN from scratch (2000 training samples) CNN from scratch (optimal sample size) Pretrained model (1000 samples) Pretrained model fine-tuned

The table shows the performance results in terms of accuracy of models trained on various sample size and architectures; CNN from Scratch and Pretrained CNN for the Cats Vs Dogs dataset. As it can be seen all models had a test accuracy of 1.0 which means that the models correctly classified all the test samples, this was regardless whether the starting point was a random architecture or a pretrained one. Nonetheless, the test loss values are varying across the models that indicate how close the model's predictions are to the actual labels. It incidentally means that the lower the loss, the better the model because we want to minimize the error irrespective of the number. Specifically, for both 1000 samples and 2000 samples, the CNN trained from scratch have the lowest loss for all the settings, and it provides a better generalization than the pretrained CNN, which, although achieving 100 percent accuracy, has a relatively higher loss in all the cases, especially for the fine-tune CNN. This means that even though all models learn well, the models trained from scratch are better at generalization according to their lower loss values.

```
import matplotlib.pyplot as plt

# Training and validation accuracy
def plot_training_history(history, title):
    acc = history.history['accuracy']
    loss = history.history['loss']

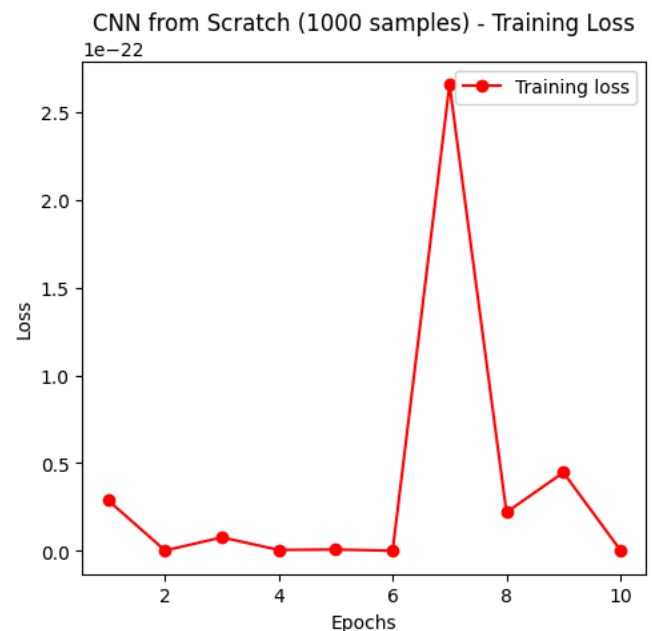
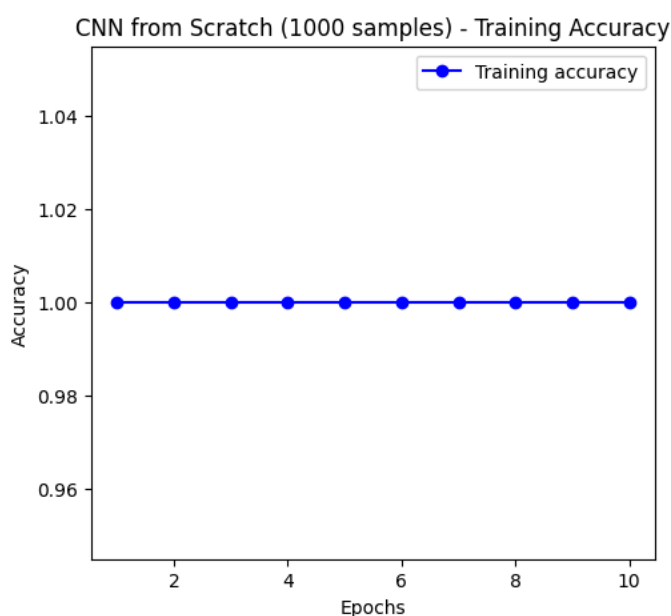
    epochs = range(1, len(acc) + 1)

    # Plot accuracy
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(epochs, acc, 'bo-', label='Training accuracy')
    plt.title(f'{title} - Training Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
```

```
# Plot loss
plt.subplot(1, 2, 2)
plt.plot(epochs, loss, 'ro-', label='Training loss')
plt.title(f'{title} - Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()

# Plot history for CNN from scratch (1000 samples)
plot_training_history(history, "CNN from Scratch (1000 samples)")
```



The given graphs present the training accuracy and the training loss for the CNN model trained from the scratch on 1000 samples. The accuracy graph (left) shows that the model had achieved the maximum possible accuracy of 1.00 for all the 10 epochs which means that the model was able to classify all the training samples in the first epoch and maintained the early high accuracy till the tenth epoch. However, the loss graph (right) indicates some fluctuations in the loss; starting with a low loss, which then jumps immensely at epoch 6 and then rapidly falls to near-zero losses. This makes sense, since while the model maintained a level of accuracy, there are slight differences in

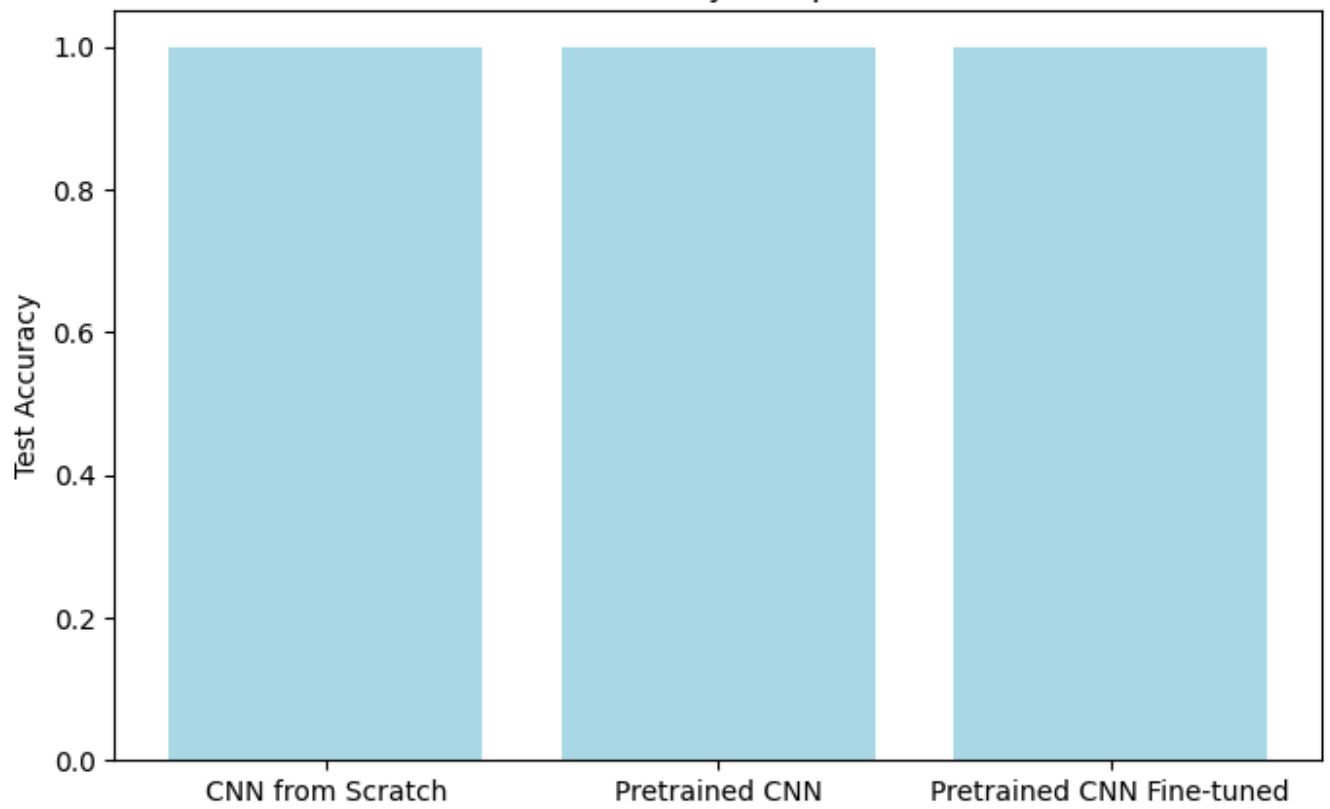
the methods the model used to fit the given predictions to the labels at some points in the training process. Consequently, the increase of the loss indicates fact that the error could increase for some period of time and only stored information influence the result, but observing the overall loss in the bottom of the picture, we can conclude that it is very low and the model has provided the excellent fit to the data.

```
# Plot comparison of test accuracy
plt.figure(figsize=(8, 5))
plt.bar(results['Model'], results['Test Accuracy'], color='lightblue')
plt.title('Test Accuracy Comparison')
plt.ylabel('Test Accuracy')
plt.show()

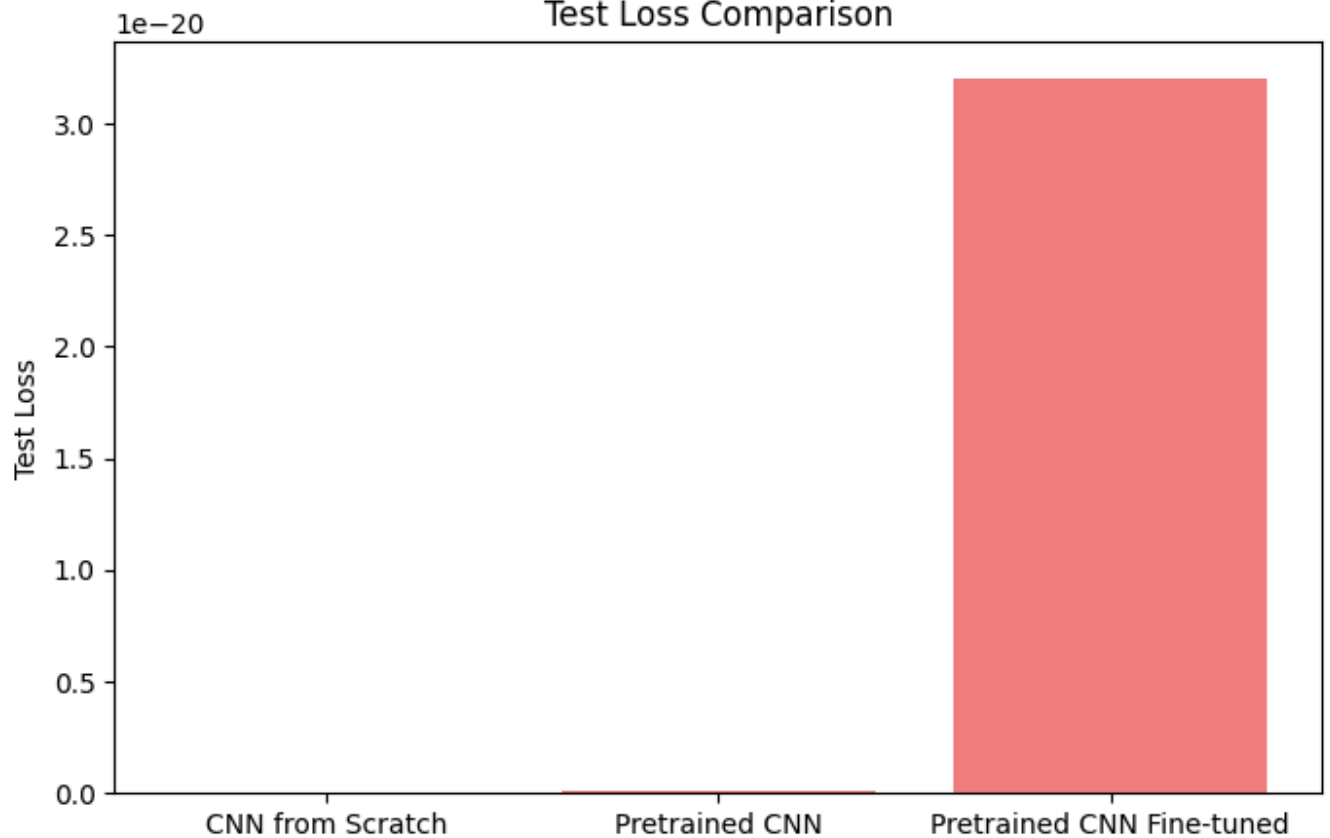
# Plot comparison of test loss
plt.figure(figsize=(8, 5))
plt.bar(results['Model'], results['Test Loss'], color='lightcoral')
plt.title('Test Loss Comparison')
plt.ylabel('Test Loss')
plt.show()
```



Test Accuracy Comparison



Test Loss Comparison



Relationship between training sample size and choice of network

This is specifically centered on the balance between the sample size used in training and the option of using a CNN from the raw data or using a pretrained model. **Training from Scratch** With small number of samples (for example 1000 samples) there is a problem of overfitting resulting from training CNN from scratch, as the model has fewer instances to train from, and hence performs poorly on unseen data. This often requires methods like data augmentation or regularization to avoid it. If trying out a model with a large sample size (for example, from 1000 to 2000), it is obvious that the model is trained with a greater variety of examples which would be beneficial for its performance on other data sets. This in turn yields better performance, less loss and stability over epochs which is very important. **Pretrained Networks** The pretrained weights are learned from a large number of images in a dataset for example ImageNet, and are better to use when you have a small amount of samples in your dataset such as 1000 samples. Despite the small training data set the pretrained network is able to harness the learned features consequently attaining a high accuracy as well as low loss. As sample size grows, previous work performs well but may need some adjustment to work optimally for the specific dataset. Training the model on the entirely different new task also prevalent in fine-tuning, as it allows to maximize the dimensions that are optimal for the new task and maintain all the obtained weights but the necessary to be improved.