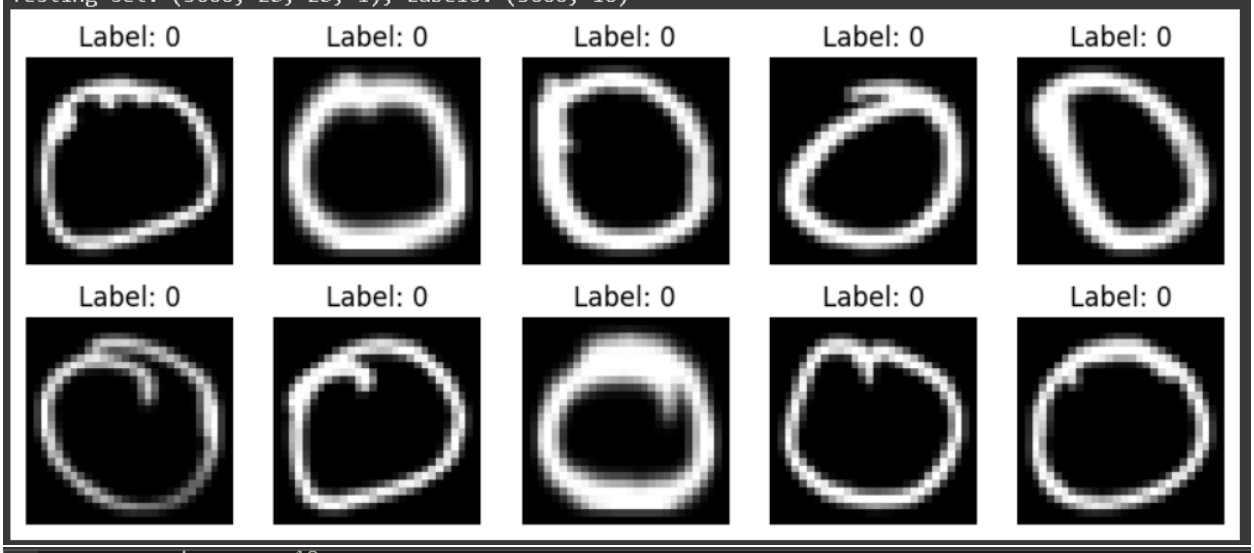
Worksheet4

Sgd Optimizer Model

```python
# Define dataset paths
train_dir = "/content/drive/MyDrive/Worksheet4_AI/Dataset/Train"
test_dir = "/content/drive/MyDrive/Worksheet4_AI/Dataset/Test"

# Define image size
img_height, img_width = 28, 28

# Function to load images and labels using PIL
def load_images_from_folder(folder):
    images = []
    labels = []
    class_names = sorted(os.listdir(folder))  # Sorted class names (digit_0, digit_1, ...)
    class_map = {name: i for i, name in enumerate(class_names)}  # Map class names to labels

    for class_name in class_names:
        class_path = os.path.join(folder, class_name)
        label = class_map[class_name]

        for filename in os.listdir(class_path):
            img_path = os.path.join(class_path, filename)
            # Load image using PIL
            img = Image.open(img_path).convert("L")  # Convert to grayscale
            img = img.resize((img_width, img_height))  # Resize to (28,28)
            img = np.array(img) / 255.0  # Normalize pixel values to [0,1]
```

```python
            images.append(img)
            labels.append(label)

    return np.array(images), np.array(labels)

# Load training and testing datasets
x_train, y_train = load_images_from_folder(train_dir)
x_test, y_test = load_images_from_folder(test_dir)

# Reshape images for Keras input
x_train = x_train.reshape(-1, img_height, img_width, 1)  # Shape (num_samples, 28, 28, 1)
x_test = x_test.reshape(-1, img_height, img_width, 1)

# One-hot encode labels
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)

# Print dataset shape
print(f"Training set: {x_train.shape}, Labels: {y_train.shape}")
print(f"Testing set: {x_test.shape}, Labels: {y_test.shape}")

# Visualize some images
plt.figure(figsize=(10, 4))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(x_train[i].reshape(28, 28), cmap='gray')  # Fixed incorrect quotes
    plt.title(f"Label: {np.argmax(y_train[i])}")
    plt.axis("off")
plt.show()
```

```
Training set: (17000, 28, 28, 1), Labels: (17000, 10)
Testing set: (3000, 28, 28, 1), Labels: (3000, 10)
```




```
num_classes = 10
input_shape = (28*28, 1)
model = keras.Sequential(
[
keras.layers.Input(shape=input_shape),
keras.layers.Flatten(),
keras.layers.Dense(64, activation="sigmoid"),
keras.layers.Dense(128, activation="sigmoid"),
keras.layers.Dense(256, activation="sigmoid"),
keras.layers.Dense(num_classes, activation="softmax"),
]
)
```

```
model.summary()
```

Model: "sequential_8"

Layer (type)	Output Shape	Param #
flatten_8 (Flatten)	(None, 784)	0
dense_26 (Dense)	(None, 64)	50,240
dense_27 (Dense)	(None, 128)	8,320
dense_28 (Dense)	(None, 256)	33,024
dense_29 (Dense)	(None, 10)	2,570

Total params: 94,154 (367.79 KB)
Trainable params: 94,154 (367.79 KB)
Non-trainable params: 0 (0.00 B)

```
[ ] model.compile(
    optimizer="sgd",
    loss="categorical_crossentropy",
    metrics=["accuracy"]
    )
```

```
x_train, y_train = shuffle(x_train, y_train, random_state=42)

batch_size = 128
epochs = 100

callbacks = [
    keras.callbacks.ModelCheckpoint(filepath="model_at_epoch_{epoch}.keras"),
    keras.callbacks.EarlyStopping(monitor="val_loss", patience = 4,),
]

history = model.fit(
    x_train,
    y_train,
    batch_size=batch_size,
    epochs=epochs,
    validation_split = 0.15,
    callbacks=callbacks,
)
```

```
Epoch 1/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.6654 - loss: 1.2685 - val_accuracy: 0.6400 - val_loss: 1.2532
Epoch 2/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.6653 - loss: 1.2388 - val_accuracy: 0.6557 - val_loss: 1.2308
Epoch 3/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.6652 - loss: 1.2198 - val_accuracy: 0.6635 - val_loss: 1.2108
Epoch 4/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.6711 - loss: 1.1970 - val_accuracy: 0.6780 - val_loss: 1.1919
Epoch 5/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.6748 - loss: 1.1754 - val_accuracy: 0.6804 - val_loss: 1.1698
Epoch 6/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - accuracy: 0.6831 - loss: 1.1519 - val_accuracy: 0.7020 - val_loss: 1.1508
Epoch 7/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.7022 - loss: 1.1257 - val_accuracy: 0.6816 - val_loss: 1.1325
Epoch 8/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.6950 - loss: 1.1115 - val_accuracy: 0.6847 - val_loss: 1.1115
Epoch 9/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.7025 - loss: 1.0855 - val_accuracy: 0.6929 - val_loss: 1.0912
Epoch 10/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.7069 - loss: 1.0727 - val_accuracy: 0.7200 - val_loss: 1.0725
Epoch 11/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.7100 - loss: 1.0556 - val_accuracy: 0.7086 - val_loss: 1.0521
Epoch 12/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.7143 - loss: 1.0380 - val_accuracy: 0.7020 - val_loss: 1.0368
Epoch 13/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.7191 - loss: 1.0172 - val_accuracy: 0.7251 - val_loss: 1.0180
Epoch 14/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.7238 - loss: 0.9874 - val_accuracy: 0.7373 - val_loss: 0.9985
Epoch 15/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.7247 - loss: 0.9759 - val_accuracy: 0.7353 - val_loss: 0.9811
Epoch 16/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.7364 - loss: 0.9618 - val_accuracy: 0.7133 - val_loss: 0.9639
```

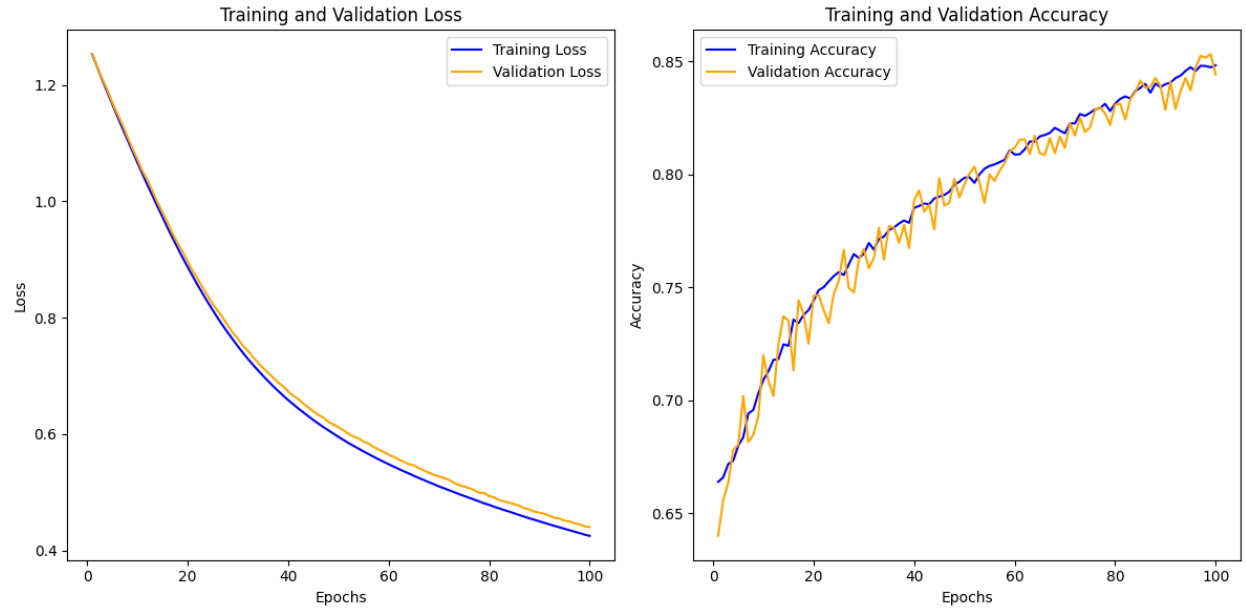
```
113/113 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - accuracy: 0.7403 - loss: 0.8886 - val_accuracy: 0.7463 - val_loss: 0.8969
Epoch 21/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.7500 - loss: 0.8689 - val_accuracy: 0.7467 - val_loss: 0.8805
Epoch 22/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.7518 - loss: 0.8572 - val_accuracy: 0.7400 - val_loss: 0.8670
Epoch 23/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.7516 - loss: 0.8433 - val_accuracy: 0.7341 - val_loss: 0.8517
Epoch 24/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.7572 - loss: 0.8200 - val_accuracy: 0.7471 - val_loss: 0.8377
Epoch 25/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.7552 - loss: 0.8201 - val_accuracy: 0.7529 - val_loss: 0.8233
Epoch 26/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.7571 - loss: 0.8087 - val_accuracy: 0.7667 - val_loss: 0.8121
Epoch 27/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.7622 - loss: 0.7933 - val_accuracy: 0.7498 - val_loss: 0.8000
Epoch 28/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.7621 - loss: 0.7767 - val_accuracy: 0.7478 - val_loss: 0.7868
Epoch 29/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.7622 - loss: 0.7612 - val_accuracy: 0.7624 - val_loss: 0.7741
Epoch 30/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.7630 - loss: 0.7530 - val_accuracy: 0.7671 - val_loss: 0.7638
Epoch 31/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.7680 - loss: 0.7452 - val_accuracy: 0.7584 - val_loss: 0.7518
Epoch 32/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.7620 - loss: 0.7366 - val_accuracy: 0.7631 - val_loss: 0.7434
Epoch 33/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.7711 - loss: 0.7183 - val_accuracy: 0.7765 - val_loss: 0.7325
Epoch 34/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.7724 - loss: 0.7080 - val_accuracy: 0.7624 - val_loss: 0.7226
Epoch 35/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.7727 - loss: 0.7040 - val_accuracy: 0.7773 - val_loss: 0.7137
Epoch 36/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.7789 - loss: 0.6986 - val_accuracy: 0.7765 - val_loss: 0.7054
```

```
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.7925 - loss: 0.6066 - val_accuracy: 0.7875 - val_loss: 0.6285
Epoch 48/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.7987 - loss: 0.5978 - val_accuracy: 0.7980 - val_loss: 0.6212
Epoch 49/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.7986 - loss: 0.6011 - val_accuracy: 0.7898 - val_loss: 0.6167
Epoch 50/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.7936 - loss: 0.6018 - val_accuracy: 0.7957 - val_loss: 0.6119
Epoch 51/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.7978 - loss: 0.5945 - val_accuracy: 0.8004 - val_loss: 0.6069
Epoch 52/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.7980 - loss: 0.5794 - val_accuracy: 0.8035 - val_loss: 0.6006
Epoch 53/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8046 - loss: 0.5808 - val_accuracy: 0.7965 - val_loss: 0.5959
Epoch 54/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8008 - loss: 0.5758 - val_accuracy: 0.7875 - val_loss: 0.5928
Epoch 55/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.7993 - loss: 0.5724 - val_accuracy: 0.8000 - val_loss: 0.5872
Epoch 56/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8081 - loss: 0.5618 - val_accuracy: 0.7973 - val_loss: 0.5836
Epoch 57/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8038 - loss: 0.5657 - val_accuracy: 0.8016 - val_loss: 0.5782
Epoch 58/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.8037 - loss: 0.5527 - val_accuracy: 0.8051 - val_loss: 0.5735
Epoch 59/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.8043 - loss: 0.5587 - val_accuracy: 0.8102 - val_loss: 0.5695
Epoch 60/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.8138 - loss: 0.5469 - val_accuracy: 0.8118 - val_loss: 0.5647
Epoch 61/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.8101 - loss: 0.5433 - val_accuracy: 0.8153 - val_loss: 0.5609
Epoch 62/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.8016 - loss: 0.5509 - val_accuracy: 0.8157 - val_loss: 0.5565
Epoch 63/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8119 - loss: 0.5401 - val_accuracy: 0.8090 - val_loss: 0.5523
```

```
Epoch 84/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8329 - loss: 0.4772 - val_accuracy: 0.8365 - val_loss: 0.4822
Epoch 85/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8409 - loss: 0.4585 - val_accuracy: 0.8416 - val_loss: 0.4796
Epoch 86/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8382 - loss: 0.4651 - val_accuracy: 0.8384 - val_loss: 0.4770
Epoch 87/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 10ms/step - accuracy: 0.8408 - loss: 0.4545 - val_accuracy: 0.8388 - val_loss: 0.4727
Epoch 88/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.8449 - loss: 0.4488 - val_accuracy: 0.8427 - val_loss: 0.4704
Epoch 89/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.8372 - loss: 0.4509 - val_accuracy: 0.8392 - val_loss: 0.4669
Epoch 90/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8413 - loss: 0.4541 - val_accuracy: 0.8286 - val_loss: 0.4651
Epoch 91/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8382 - loss: 0.4506 - val_accuracy: 0.8412 - val_loss: 0.4628
Epoch 92/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8433 - loss: 0.4445 - val_accuracy: 0.8290 - val_loss: 0.4598
Epoch 93/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8411 - loss: 0.4479 - val_accuracy: 0.8369 - val_loss: 0.4562
Epoch 94/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8466 - loss: 0.4346 - val_accuracy: 0.8427 - val_loss: 0.4550
Epoch 95/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8500 - loss: 0.4386 - val_accuracy: 0.8373 - val_loss: 0.4511
Epoch 96/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8524 - loss: 0.4248 - val_accuracy: 0.8475 - val_loss: 0.4499
Epoch 97/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8493 - loss: 0.4279 - val_accuracy: 0.8525 - val_loss: 0.4467
Epoch 98/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8442 - loss: 0.4394 - val_accuracy: 0.8518 - val_loss: 0.4446
Epoch 99/100
113/113 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.8483 - loss: 0.4290 - val_accuracy: 0.8533 - val_loss: 0.4417
Epoch 100/100
```

```python
import matplotlib.pyplot as plt

# Assuming 'history' is the object returned by model.fit()
# Extracting training and validation loss
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# Extracting training and validation accuracy (if metrics were specified)
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

# Plotting training and validation loss
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(range(1, len(train_loss) + 1), train_loss, label='Training Loss', color='blue')
plt.plot(range(1, len(val_loss) + 1), val_loss, label='Validation Loss', color='orange')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()

# Plotting training and validation accuracy
plt.subplot(1, 2, 2)
plt.plot(range(1, len(train_acc) + 1), train_acc, label='Training Accuracy', color='blue')
plt.plot(range(1, len(val_acc) + 1), val_acc, label='Validation Accuracy', color='orange')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
```

```
plt.legend()

plt.tight_layout()
plt.show()
```

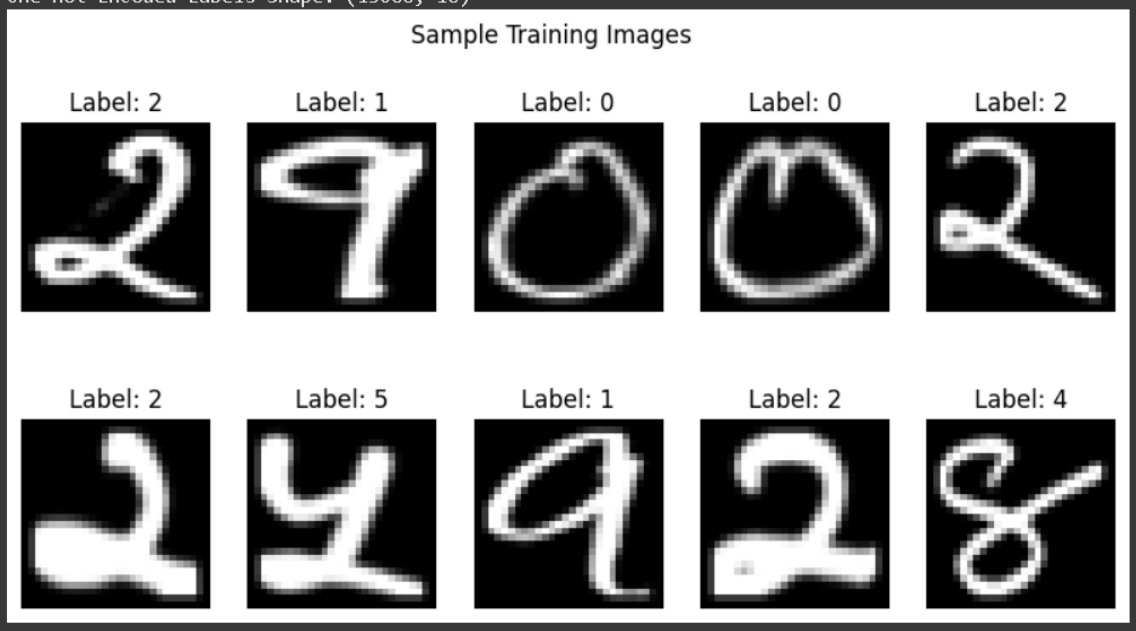



```
[ ] test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
    print(f"Test Accuracy: {test_acc:.4f}")

94/94 - 0s - 2ms/step - accuracy: 0.8493 - loss: 0.4373
Test Accuracy: 0.8493

[ ] model.save("devnagari_digit_classifier.h5")

    loaded_model = tf.keras.models.load_model("devnagari_digit_classifier.h5")

    test_loss, test_acc = loaded_model.evaluate(x_test, y_test, verbose=2)
    print(f"Loaded Model Test Accuracy: {test_acc:.4f}")

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
94/94 - 1s - 6ms/step - accuracy: 0.8493 - loss: 0.4373
Loaded Model Test Accuracy: 0.8493
```

```python
train_dir = "/content/drive/MyDrive/Worksheet4_AI/Dataset/Train"
test_dir = "/content/drive/MyDrive/Worksheet4_AI/Dataset/Test"

def load_images_from_folder(folder):
    images, labels = [], []
    classes = sorted(os.listdir(folder))
    class_map = {class_name: i for i, class_name in enumerate(classes)}

    for class_name in classes:
        class_folder = os.path.join(folder, class_name)
        if not os.path.isdir(class_folder):
            continue

        for image_name in os.listdir(class_folder):
            image_path = os.path.join(class_folder, image_name)
            try:
                img = Image.open(image_path).convert('L')
                img = img.resize((28, 28))
                img = np.array(img) / 255.0
                images.append(img)
                labels.append(class_map[class_name])
            except Exception as e:
                print(f"Error loading image {image_path}: {e}")

    return np.array(images), np.array(labels)

x_train, y_train = load_images_from_folder(train_dir)
x_test, y_test = load_images_from_folder(test_dir)
```

```python
x_train = x_train.reshape(x_train.shape[0], 28*28)
x_test = x_test.reshape(x_test.shape[0], 28*28)

y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)

x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, random_state=42)

print(f"Training Data Shape: {x_train.shape}, Validation Shape: {x_val.shape}, Test Shape: {x_test.shape}")
print(f"One-hot Encoded Labels Shape: {y_train.shape}")

fig, axes = plt.subplots(2, 5, figsize=(10, 5))
for i, ax in enumerate(axes.flat):
    ax.imshow(x_train[i].reshape(28, 28), cmap="gray")
    ax.set_title(f"Label: {np.argmax(y_train[i])}")
    ax.axis("off")

plt.suptitle("Sample Training Images")
plt.show()
```

```
Training Data Shape: (13600, 784), Validation Shape: (3400, 784), Test Shape: (3000, 784)
One-hot Encoded Labels Shape: (13600, 10)
```



Task 2 - Building Fully Connected Neural Network Model

```
model = Sequential([
    Dense(64, activation='sigmoid', input_shape=(28*28,)),
    Dense(128, activation='sigmoid'),
    Dense(256, activation='sigmoid'),
    Dense(10, activation='softmax')
])

model.summary()
```

Model: "sequential_11"

Layer (type)	Output Shape	Param #
dense_38 (Dense)	(None, 64)	50,240
dense_39 (Dense)	(None, 128)	8,320
dense_40 (Dense)	(None, 256)	33,024
dense_41 (Dense)	(None, 10)	2,570

Total params: 94,154 (367.79 KB)
Trainable params: 94,154 (367.79 KB)
Non-trainable params: 0 (0.00 B)

Task 3 - Compiling the Model

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

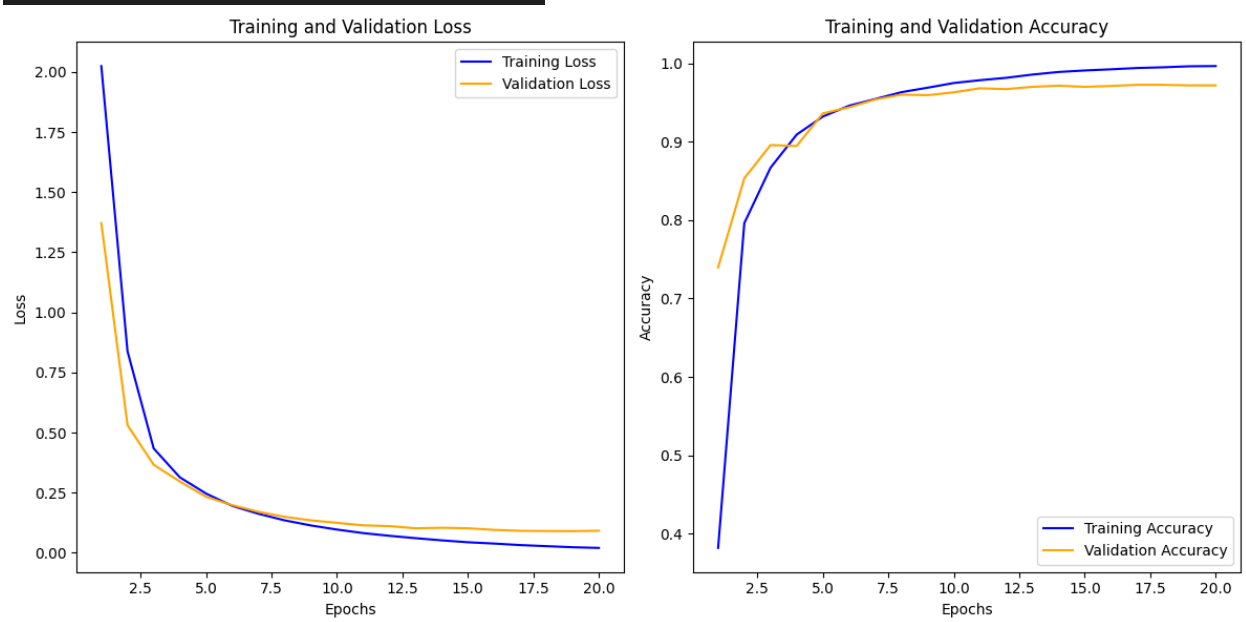
Task 4 - Train the Model

```
callbacks = [
    keras.callbacks.ModelCheckpoint(filepath="model_at_epoch_{epoch}.keras", save_best_only=True, monitor="val_loss"),
    keras.callbacks.EarlyStopping(monitor="val_loss", patience = 4, restore_best_weights=True)
]

# Train the model
history = model.fit(
    x_train, y_train,
    batch_size=128,
    epochs=20,
    validation_split=0.2,
    callbacks=callbacks
)
```

```
Epoch 1/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - accuracy: 0.2127 - loss: 2.2424 - val_accuracy: 0.7397 - val_loss: 1.3718
Epoch 2/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.7700 - loss: 1.0598 - val_accuracy: 0.8533 - val_loss: 0.5311
Epoch 3/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.8625 - loss: 0.4657 - val_accuracy: 0.8956 - val_loss: 0.3664
Epoch 4/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - accuracy: 0.9023 - loss: 0.3371 - val_accuracy: 0.8945 - val_loss: 0.2967
Epoch 5/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - accuracy: 0.9271 - loss: 0.2592 - val_accuracy: 0.9360 - val_loss: 0.2332
Epoch 6/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9412 - loss: 0.2048 - val_accuracy: 0.9438 - val_loss: 0.1983
Epoch 7/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9541 - loss: 0.1634 - val_accuracy: 0.9537 - val_loss: 0.1706
Epoch 8/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9595 - loss: 0.1449 - val_accuracy: 0.9599 - val_loss: 0.1500
Epoch 9/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9661 - loss: 0.1205 - val_accuracy: 0.9592 - val_loss: 0.1351
Epoch 10/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9766 - loss: 0.0954 - val_accuracy: 0.9629 - val_loss: 0.1247
Epoch 11/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9797 - loss: 0.0814 - val_accuracy: 0.9680 - val_loss: 0.1144
Epoch 12/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9808 - loss: 0.0727 - val_accuracy: 0.9669 - val_loss: 0.1112
Epoch 13/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9873 - loss: 0.0594 - val_accuracy: 0.9699 - val_loss: 0.1021
Epoch 14/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9899 - loss: 0.0505 - val_accuracy: 0.9713 - val_loss: 0.1041
Epoch 15/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9924 - loss: 0.0418 - val_accuracy: 0.9699 - val_loss: 0.1021
Epoch 16/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9917 - loss: 0.0394 - val_accuracy: 0.9710 - val_loss: 0.0955
Epoch 17/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - accuracy: 0.9942 - loss: 0.0327 - val_accuracy: 0.9724 - val_loss: 0.0915
Epoch 18/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - accuracy: 0.9949 - loss: 0.0264 - val_accuracy: 0.9724 - val_loss: 0.0907
Epoch 19/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - accuracy: 0.9966 - loss: 0.0240 - val_accuracy: 0.9717 - val_loss: 0.0902
Epoch 20/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9968 - loss: 0.0204 - val_accuracy: 0.9717 - val_loss: 0.0917
```

```python
# Assuming 'history' is the object returned by model.fit()
# Extracting training and validation loss
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# Extracting training and validation accuracy (if metrics were specified)
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

# Plotting training and validation loss
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.plot(range(1, len(train_loss) + 1), train_loss, label='Training Loss', color='blue')
plt.plot(range(1, len(val_loss) + 1), val_loss, label='Validation Loss', color='orange')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()

# Plotting training and validation accuracy
plt.subplot(1, 2, 2)
plt.plot(range(1, len(train_acc) + 1), train_acc, label='Training Accuracy', color='blue')
plt.plot(range(1, len(val_acc) + 1), val_acc, label='Validation Accuracy', color='orange')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
```

```python
plt.tight_layout()
plt.show()
```



Task 5 - Evaluate the Model

```
[ ] test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
    print(f"Test Accuracy: {test_acc:.4f}")
    print(f"Test Loss: {test_loss:.4f}")
```

```
94/94 - 0s - 2ms/step - accuracy: 0.9740 - loss: 0.0903
Test Accuracy: 0.9740
Test Loss: 0.0903
```

Task 6 - Saving and Loading the Model

```
model.save("devnagari_digit_classifier.h5")
print("Model saved successfully as 'devnagari_digit_classifier.h5'!")

loaded_model = tf.keras.models.load_model("devnagari_digit_classifier.h5")
print("Model loaded successfully!")

test_loss, test_acc = loaded_model.evaluate(x_test, y_test, verbose=2)
print(f"Loaded Model Test Accuracy: {test_acc:.4f}")
print(f"Loaded Model Test Loss: {test_loss:.4f}")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
Model saved successfully as 'devnagari_digit_classifier.h5'!
Model loaded successfully!
94/94 - 1s - 6ms/step - accuracy: 0.9740 - loss: 0.0903
Loaded Model Test Accuracy: 0.9740
Loaded Model Test Loss: 0.0903
```

Task 7 - Making Predictions

```
num_samples = 5
random_indices = np.random.choice(len(x_test), num_samples, replace=False)
sample_images = x_test[random_indices]
sample_labels = y_test[random_indices]

predictions = loaded_model.predict(sample_images)

predicted_labels = np.argmax(predictions, axis=1)
true_labels = np.argmax(sample_labels, axis=1)

plt.figure(figsize=(10, 5))
for i in range(num_samples):
    plt.subplot(1, num_samples, i + 1)
    plt.imshow(sample_images[i].reshape(28, 28), cmap="gray")
    plt.title(f"Pred: {predicted_labels[i]}\nTrue: {true_labels[i]}")
    plt.axis("off")

plt.suptitle("Model Predictions on Test Images")
plt.show()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step
```