LeNet on MNIST Dataset

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, AveragePooling2D, Flatten, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import CategoricalCrossentropy

# Load and preprocess the MNIST dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Reshape and normalize the images
train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255

# Convert labels to categorical one-hot encoding
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# Define the LeNet model architecture
def LeNet():
    model = Sequential([
        Conv2D(6, kernel_size=(5, 5), activation='relu', input_shape=(28, 28, 1)),
        AveragePooling2D(),
        Conv2D(16, kernel_size=(5, 5), activation='relu'),
        AveragePooling2D(),
        Flatten(),
        Dense(120, activation='relu'),
        Dense(84, activation='relu'),
        Dense(10, activation='softmax')
    ])
    return model

# Compile the model
model = LeNet()
model.compile(optimizer=Adam(),
              loss=CategoricalCrossentropy(),
              metrics=['accuracy'])

# Train the model
history = model.fit(train_images, train_labels, epochs=10, batch_size=128, validation_split=0.2)

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc}')

# Save the results
results = {
    'test_loss': test_loss,
    'test_accuracy': test_acc,
    'history': history.history
}

import json
with open('lenet_mnist_results.json', 'w') as f:
    json.dump(results, f)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 2s 0us/step
Epoch 1/10
375/375 [==============================] - 8s 8ms/step - loss: 0.4548 - accuracy: 0.8639 - val_loss: 0.1538 - val_accuracy: 0.9546
Epoch 2/10
375/375 [==============================] - 2s 4ms/step - loss: 0.1308 - accuracy: 0.9598 - val_loss: 0.1076 - val_accuracy: 0.9668
Epoch 3/10
375/375 [==============================] - 2s 4ms/step - loss: 0.0890 - accuracy: 0.9723 - val_loss: 0.0801 - val_accuracy: 0.9768
Epoch 4/10
375/375 [==============================] - 2s 4ms/step - loss: 0.0673 - accuracy: 0.9796 - val_loss: 0.0625 - val_accuracy: 0.9816
Epoch 5/10
375/375 [==============================] - 2s 4ms/step - loss: 0.0546 - accuracy: 0.9827 - val_loss: 0.0538 - val_accuracy: 0.9834
Epoch 6/10
375/375 [==============================] - 2s 4ms/step - loss: 0.0486 - accuracy: 0.9851 - val_loss: 0.0646 - val_accuracy: 0.9808
Epoch 7/10
375/375 [==============================] - 2s 5ms/step - loss: 0.0434 - accuracy: 0.9864 - val_loss: 0.0605 - val_accuracy: 0.9818
Epoch 8/10
375/375 [==============================] - 2s 6ms/step - loss: 0.0373 - accuracy: 0.9882 - val_loss: 0.0525 - val_accuracy: 0.9849
Epoch 9/10
375/375 [==============================] - 2s 4ms/step - loss: 0.0324 - accuracy: 0.9897 - val_loss: 0.0462 - val_accuracy: 0.9862
Epoch 10/10
375/375 [==============================] - 2s 4ms/step - loss: 0.0301 - accuracy: 0.9906 - val_loss: 0.0487 - val_accuracy: 0.9855
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.0390 - accuracy: 0.9876
Test accuracy: 0.9876000285148621
```

LeNet Fashion MNIST

```python
import tensorflow as tf
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, AveragePooling2D, Flatten, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import CategoricalCrossentropy

# Load and preprocess the Fashion MNIST dataset
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

# Reshape and normalize the images
train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255

# Convert labels to categorical one-hot encoding
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# Define the LeNet model architecture
def LeNet():
    model = Sequential([
        Conv2D(6, kernel_size=(5, 5), activation='relu', input_shape=(28, 28, 1)),
        AveragePooling2D(),
        Conv2D(16, kernel_size=(5, 5), activation='relu'),
        AveragePooling2D(),
        Flatten(),
        Dense(120, activation='relu'),
        Dense(84, activation='relu'),
        Dense(10, activation='softmax')
    ])
    return model

# Compile the model
model = LeNet()
model.compile(optimizer=Adam(),
              loss=CategoricalCrossentropy(),
              metrics=['accuracy'])

# Train the model
history = model.fit(train_images, train_labels, epochs=10, batch_size=128, validation_split=0.2)

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc}')

# Save the results
results = {
    'test_loss': test_loss,
    'test_accuracy': test_acc,
    'history': history.history
}

import json
with open('lenet_fashion_mnist_results.json', 'w') as f:
    json.dump(results, f)
```

```
Epoch 1/10
375/375 [==============================] - 3s 5ms/step - loss: 0.7622 - accuracy: 0.7190 - val_loss: 0.5464 - val_accuracy: 0.7990
Epoch 2/10
375/375 [==============================] - 2s 6ms/step - loss: 0.5077 - accuracy: 0.8157 - val_loss: 0.5066 - val_accuracy: 0.8091
Epoch 3/10
375/375 [==============================] - 2s 5ms/step - loss: 0.4405 - accuracy: 0.8412 - val_loss: 0.4274 - val_accuracy: 0.8438
Epoch 4/10
375/375 [==============================] - 2s 4ms/step - loss: 0.3978 - accuracy: 0.8564 - val_loss: 0.3940 - val_accuracy: 0.8567
Epoch 5/10
375/375 [==============================] - 2s 4ms/step - loss: 0.3736 - accuracy: 0.8644 - val_loss: 0.3722 - val_accuracy: 0.8673
Epoch 6/10
375/375 [==============================] - 2s 4ms/step - loss: 0.3515 - accuracy: 0.8717 - val_loss: 0.3729 - val_accuracy: 0.8668
Epoch 7/10
375/375 [==============================] - 2s 5ms/step - loss: 0.3330 - accuracy: 0.8785 - val_loss: 0.3584 - val_accuracy: 0.8707
Epoch 8/10
375/375 [==============================] - 2s 4ms/step - loss: 0.3204 - accuracy: 0.8822 - val_loss: 0.3472 - val_accuracy: 0.8749
Epoch 9/10
375/375 [==============================] - 2s 5ms/step - loss: 0.3092 - accuracy: 0.8872 - val_loss: 0.3292 - val_accuracy: 0.8799
Epoch 10/10
375/375 [==============================] - 2s 5ms/step - loss: 0.3010 - accuracy: 0.8902 - val_loss: 0.3389 - val_accuracy: 0.8748
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.3536 - accuracy: 0.8673
Test accuracy: 0.8672999739646912
```

LeNet on CIFAR10

```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, AveragePooling2D, Flatten, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import CategoricalCrossentropy

# Load and preprocess the CIFAR-10 dataset
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()

# Normalize the images
train_images = train_images.astype('float32') / 255
test_images = test_images.astype('float32') / 255

# Convert labels to categorical one-hot encoding
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# Define the LeNet model architecture
def LeNet():
    model = Sequential([
        Conv2D(6, kernel_size=(5, 5), activation='relu', input_shape=(32, 32, 3)),
        AveragePooling2D(),
        Conv2D(16, kernel_size=(5, 5), activation='relu'),
        AveragePooling2D(),
        Flatten(),
        Dense(120, activation='relu'),
        Dense(84, activation='relu'),
        Dense(10, activation='softmax')
    ])
    return model

# Compile the model
model = LeNet()
model.compile(optimizer=Adam(),
              loss=CategoricalCrossentropy(),
              metrics=['accuracy'])

# Train the model
history = model.fit(train_images, train_labels, epochs=20, batch_size=128, validation_split=0.2)

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc}')

# Save the results
results = {
    'test_loss': test_loss,
    'test_accuracy': test_acc,
    'history': history.history
}

import json
with open('lenet_cifar10_results.json', 'w') as f:
    json.dump(results, f)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 13s 0us/step
Epoch 1/20
313/313 [==============================] - 4s 7ms/step - loss: 1.8498 - accuracy: 0.3273 - val_loss: 1.6244 - val_accuracy: 0.4200
Epoch 2/20
313/313 [==============================] - 2s 5ms/step - loss: 1.5516 - accuracy: 0.4442 - val_loss: 1.5059 - val_accuracy: 0.4633
Epoch 3/20
313/313 [==============================] - 2s 5ms/step - loss: 1.4382 - accuracy: 0.4803 - val_loss: 1.4229 - val_accuracy: 0.4865
Epoch 4/20
313/313 [==============================] - 2s 5ms/step - loss: 1.3680 - accuracy: 0.5083 - val_loss: 1.3606 - val_accuracy: 0.5150
Epoch 5/20
313/313 [==============================] - 2s 5ms/step - loss: 1.3046 - accuracy: 0.5342 - val_loss: 1.3248 - val_accuracy: 0.5257
Epoch 6/20
313/313 [==============================] - 2s 7ms/step - loss: 1.2544 - accuracy: 0.5520 - val_loss: 1.2897 - val_accuracy: 0.5364
Epoch 7/20
313/313 [==============================] - 2s 6ms/step - loss: 1.2175 - accuracy: 0.5627 - val_loss: 1.2632 - val_accuracy: 0.5581
Epoch 8/20
313/313 [==============================] - 2s 5ms/step - loss: 1.1682 - accuracy: 0.5838 - val_loss: 1.2408 - val_accuracy: 0.5621
Epoch 9/20
313/313 [==============================] - 2s 5ms/step - loss: 1.1380 - accuracy: 0.5951 - val_loss: 1.2276 - val_accuracy: 0.5688
```

```
Epoch 10/20
313/313 [==============================] - 2s 5ms/step - loss: 1.1106 - accuracy: 0.6046 - val_loss: 1.2186 - val_accuracy: 0.5682
Epoch 11/20
313/313 [==============================] - 2s 5ms/step - loss: 1.0767 - accuracy: 0.6187 - val_loss: 1.1895 - val_accuracy: 0.5862
Epoch 12/20
313/313 [==============================] - 2s 5ms/step - loss: 1.0476 - accuracy: 0.6257 - val_loss: 1.2206 - val_accuracy: 0.5793
Epoch 13/20
313/313 [==============================] - 2s 6ms/step - loss: 1.0247 - accuracy: 0.6352 - val_loss: 1.1869 - val_accuracy: 0.5920
Epoch 14/20
313/313 [==============================] - 2s 7ms/step - loss: 1.0027 - accuracy: 0.6448 - val_loss: 1.1742 - val_accuracy: 0.5972
Epoch 15/20
313/313 [==============================] - 2s 7ms/step - loss: 0.9706 - accuracy: 0.6574 - val_loss: 1.1780 - val_accuracy: 0.5973
Epoch 16/20
313/313 [==============================] - 2s 5ms/step - loss: 0.9490 - accuracy: 0.6658 - val_loss: 1.1817 - val_accuracy: 0.5989
Epoch 17/20
313/313 [==============================] - 2s 5ms/step - loss: 0.9249 - accuracy: 0.6712 - val_loss: 1.1996 - val_accuracy: 0.5886
Epoch 18/20
313/313 [==============================] - 2s 5ms/step - loss: 0.9068 - accuracy: 0.6799 - val_loss: 1.1955 - val_accuracy: 0.5968
Epoch 19/20
313/313 [==============================] - 2s 5ms/step - loss: 0.8879 - accuracy: 0.6852 - val_loss: 1.1799 - val_accuracy: 0.6030
Epoch 20/20
313/313 [==============================] - 2s 7ms/step - loss: 0.8627 - accuracy: 0.6959 - val_loss: 1.1798 - val_accuracy: 0.6012
313/313 [==============================] - 1s 2ms/step - loss: 1.2014 - accuracy: 0.5894
Test accuracy: 0.5893999934196472
```

LeNet on CIFAR100

```
import tensorflow as tf
from tensorflow.keras.datasets import cifar100
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, AveragePooling2D, Flatten, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import CategoricalCrossentropy

# Load and preprocess the CIFAR-100 dataset
(train_images, train_labels), (test_images, test_labels) = cifar100.load_data()

# Normalize the images
train_images = train_images.astype('float32') / 255
test_images = test_images.astype('float32') / 255

# Convert labels to categorical one-hot encoding
train_labels = to_categorical(train_labels, 100)
test_labels = to_categorical(test_labels, 100)

# Define the LeNet model architecture
def LeNet():
    model = Sequential([
        Conv2D(6, kernel_size=(5, 5), activation='relu', input_shape=(32, 32, 3)),
        AveragePooling2D(),
        Conv2D(16, kernel_size=(5, 5), activation='relu'),
        AveragePooling2D(),
        Flatten(),
        Dense(120, activation='relu'),
        Dense(84, activation='relu'),
        Dense(100, activation='softmax')
    ])
    return model

# Compile the model
model = LeNet()
model.compile(optimizer=Adam(),
              loss=CategoricalCrossentropy(),
              metrics=['accuracy'])

# Train the model
history = model.fit(train_images, train_labels, epochs=20, batch_size=128, validation_split=0.2)

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc}')

# Save the results
results = {
    'test_loss': test_loss,
    'test_accuracy': test_acc,
    'history': history.history
}

import json
with open('lenet_cifar100_results.json', 'w') as f:
    json.dump(results, f)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz
169001437/169001437 [==============================] - 13s 0us/step
Epoch 1/20
313/313 [==============================] - 4s 7ms/step - loss: 4.1800 - accuracy: 0.0632 - val_loss: 3.9056 - val_accuracy: 0.1048
Epoch 2/20
313/313 [==============================] - 2s 7ms/step - loss: 3.7796 - accuracy: 0.1234 - val_loss: 3.7158 - val_accuracy: 0.1377
Epoch 3/20
313/313 [==============================] - 2s 6ms/step - loss: 3.6082 - accuracy: 0.1533 - val_loss: 3.5740 - val_accuracy: 0.1625
Epoch 4/20
313/313 [==============================] - 2s 6ms/step - loss: 3.4799 - accuracy: 0.1760 - val_loss: 3.4735 - val_accuracy: 0.1775
Epoch 5/20
313/313 [==============================] - 2s 5ms/step - loss: 3.3660 - accuracy: 0.1978 - val_loss: 3.4108 - val_accuracy: 0.1892
Epoch 6/20
313/313 [==============================] - 2s 5ms/step - loss: 3.2828 - accuracy: 0.2139 - val_loss: 3.3548 - val_accuracy: 0.2023
Epoch 7/20
313/313 [==============================] - 2s 5ms/step - loss: 3.2142 - accuracy: 0.2230 - val_loss: 3.3034 - val_accuracy: 0.2117
Epoch 8/20
313/313 [==============================] - 2s 5ms/step - loss: 3.1569 - accuracy: 0.2330 - val_loss: 3.2815 - val_accuracy: 0.2168
Epoch 9/20
313/313 [==============================] - 3s 8ms/step - loss: 3.1008 - accuracy: 0.2446 - val_loss: 3.2403 - val_accuracy: 0.2238
Epoch 10/20
313/313 [==============================] - 2s 7ms/step - loss: 3.0530 - accuracy: 0.2550 - val_loss: 3.2135 - val_accuracy: 0.2302
Epoch 11/20
313/313 [==============================] - 2s 5ms/step - loss: 3.0095 - accuracy: 0.2629 - val_loss: 3.2069 - val_accuracy: 0.2291
Epoch 12/20
313/313 [==============================] - 2s 5ms/step - loss: 2.9828 - accuracy: 0.2693 - val_loss: 3.1788 - val_accuracy: 0.2394
Epoch 13/20
```

```
313/313 [==============================] - 2s 6ms/step - loss: 2.9377 - accuracy: 0.2740 - val_loss: 3.1654 - val_accuracy: 0.2392
Epoch 14/20
313/313 [==============================] - 2s 5ms/step - loss: 2.9033 - accuracy: 0.2833 - val_loss: 3.1356 - val_accuracy: 0.2419
Epoch 15/20
313/313 [==============================] - 2s 6ms/step - loss: 2.8712 - accuracy: 0.2892 - val_loss: 3.1432 - val_accuracy: 0.2466
Epoch 16/20
313/313 [==============================] - 2s 7ms/step - loss: 2.8380 - accuracy: 0.2930 - val_loss: 3.1291 - val_accuracy: 0.2478
Epoch 17/20
313/313 [==============================] - 2s 6ms/step - loss: 2.8095 - accuracy: 0.2982 - val_loss: 3.1313 - val_accuracy: 0.2476
Epoch 18/20
313/313 [==============================] - 2s 5ms/step - loss: 2.7825 - accuracy: 0.3069 - val_loss: 3.1407 - val_accuracy: 0.2504
Epoch 19/20
313/313 [==============================] - 2s 5ms/step - loss: 2.7498 - accuracy: 0.3114 - val_loss: 3.1106 - val_accuracy: 0.2538
Epoch 20/20
313/313 [==============================] - 2s 5ms/step - loss: 2.7240 - accuracy: 0.3162 - val_loss: 3.1048 - val_accuracy: 0.2542
313/313 [==============================] - 1s 3ms/step - loss: 3.0891 - accuracy: 0.2578
Test accuracy: 0.25780001282691956
```

VGG on MNIST

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import CategoricalCrossentropy

# Load and preprocess the MNIST dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Reshape and normalize the images
train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255

# Convert labels to categorical one-hot encoding
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# Define the VGG-style model architecture
def VGG():
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(28, 28, 1)),
        Conv2D(32, (3, 3), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Conv2D(64, (3, 3), activation='relu', padding='same'),
        Conv2D(64, (3, 3), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Conv2D(128, (3, 3), activation='relu', padding='same'),
        Conv2D(128, (3, 3), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Flatten(),
        Dense(512, activation='relu'),
        Dropout(0.5),
        Dense(10, activation='softmax')
    ])
    return model

# Compile the model
model = VGG()
model.compile(optimizer=Adam(),
              loss=CategoricalCrossentropy(),
              metrics=['accuracy'])

# Train the model
history = model.fit(train_images, train_labels, epochs=20, batch_size=128, validation_split=0.2)

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc}')

# Save the results
results = {
    'test_loss': test_loss,
    'test_accuracy': test_acc,
    'history': history.history
}
```

```
}

import json
with open('vgg_mnist_results.json', 'w') as f:
    json.dump(results, f)
```

```
Epoch 1/20
375/375 [==============================] - 10s 15ms/step - loss: 0.3030 - accuracy: 0.8999 - val_loss: 0.0599 - val_accuracy: 0.982
Epoch 2/20
375/375 [==============================] - 5s 13ms/step - loss: 0.0744 - accuracy: 0.9772 - val_loss: 0.0422 - val_accuracy: 0.9883
Epoch 3/20
375/375 [==============================] - 5s 13ms/step - loss: 0.0521 - accuracy: 0.9837 - val_loss: 0.0426 - val_accuracy: 0.9876
Epoch 4/20
375/375 [==============================] - 5s 14ms/step - loss: 0.0431 - accuracy: 0.9866 - val_loss: 0.0297 - val_accuracy: 0.9918
Epoch 5/20
375/375 [==============================] - 5s 13ms/step - loss: 0.0355 - accuracy: 0.9890 - val_loss: 0.0270 - val_accuracy: 0.9919
Epoch 6/20
375/375 [==============================] - 5s 14ms/step - loss: 0.0315 - accuracy: 0.9900 - val_loss: 0.0250 - val_accuracy: 0.9934
Epoch 7/20
375/375 [==============================] - 5s 13ms/step - loss: 0.0276 - accuracy: 0.9913 - val_loss: 0.0276 - val_accuracy: 0.9927
Epoch 8/20
375/375 [==============================] - 5s 13ms/step - loss: 0.0272 - accuracy: 0.9914 - val_loss: 0.0293 - val_accuracy: 0.9918
Epoch 9/20
375/375 [==============================] - 5s 13ms/step - loss: 0.0247 - accuracy: 0.9920 - val_loss: 0.0289 - val_accuracy: 0.9918
Epoch 10/20
375/375 [==============================] - 5s 13ms/step - loss: 0.0214 - accuracy: 0.9933 - val_loss: 0.0306 - val_accuracy: 0.9927
Epoch 11/20
375/375 [==============================] - 5s 14ms/step - loss: 0.0208 - accuracy: 0.9935 - val_loss: 0.0297 - val_accuracy: 0.9933
Epoch 12/20
375/375 [==============================] - 5s 13ms/step - loss: 0.0185 - accuracy: 0.9944 - val_loss: 0.0252 - val_accuracy: 0.9936
Epoch 13/20
375/375 [==============================] - 5s 14ms/step - loss: 0.0187 - accuracy: 0.9941 - val_loss: 0.0289 - val_accuracy: 0.9926
Epoch 14/20
375/375 [==============================] - 5s 14ms/step - loss: 0.0176 - accuracy: 0.9948 - val_loss: 0.0268 - val_accuracy: 0.9933
Epoch 15/20
375/375 [==============================] - 5s 13ms/step - loss: 0.0173 - accuracy: 0.9949 - val_loss: 0.0247 - val_accuracy: 0.9937
Epoch 16/20
375/375 [==============================] - 5s 13ms/step - loss: 0.0165 - accuracy: 0.9949 - val_loss: 0.0265 - val_accuracy: 0.9933
Epoch 17/20
375/375 [==============================] - 5s 13ms/step - loss: 0.0167 - accuracy: 0.9947 - val_loss: 0.0282 - val_accuracy: 0.9935
Epoch 18/20
375/375 [==============================] - 5s 14ms/step - loss: 0.0148 - accuracy: 0.9952 - val_loss: 0.0274 - val_accuracy: 0.9934
Epoch 19/20
375/375 [==============================] - 5s 14ms/step - loss: 0.0150 - accuracy: 0.9954 - val_loss: 0.0246 - val_accuracy: 0.9937
Epoch 20/20
375/375 [==============================] - 5s 13ms/step - loss: 0.0126 - accuracy: 0.9960 - val_loss: 0.0244 - val_accuracy: 0.9933
313/313 [==============================] - 1s 4ms/step - loss: 0.0178 - accuracy: 0.9948
Test accuracy: 0.9947999715805054
```

VGG on Fashion-MNIST

```python
import tensorflow as tf
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import CategoricalCrossentropy

# Load and preprocess the Fashion MNIST dataset
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

# Reshape and normalize the images
train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255

# Convert labels to categorical one-hot encoding
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# Define the VGG-style model architecture
def VGG():
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(28, 28, 1)),
        Conv2D(32, (3, 3), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Conv2D(64, (3, 3), activation='relu', padding='same'),
        Conv2D(64, (3, 3), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Conv2D(128, (3, 3), activation='relu', padding='same'),
        Conv2D(128, (3, 3), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Flatten(),
        Dense(512, activation='relu'),
        Dropout(0.5),
        Dense(10, activation='softmax')
    ])
    return model

# Compile the model
model = VGG()
model.compile(optimizer=Adam(),
              loss=CategoricalCrossentropy(),
              metrics=['accuracy'])

# Train the model
history = model.fit(train_images, train_labels, epochs=20, batch_size=128, validation_split=0.2)

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc}')

# Save the results
results = {
    'test_loss': test_loss,
    'test_accuracy': test_acc,
    'history': history.history
}

import json
with open('vgg_fashion_mnist_results.json', 'w') as f:
    json.dump(results, f)
```

```
Epoch 1/20
375/375 [==============================] - 8s 14ms/step - loss: 0.6274 - accuracy: 0.7704 - val_loss: 0.3467 - val_accuracy: 0.8702
Epoch 2/20
375/375 [==============================] - 5s 13ms/step - loss: 0.3549 - accuracy: 0.8693 - val_loss: 0.2981 - val_accuracy: 0.8921
Epoch 3/20
375/375 [==============================] - 5s 14ms/step - loss: 0.2971 - accuracy: 0.8901 - val_loss: 0.2865 - val_accuracy: 0.8912
Epoch 4/20
375/375 [==============================] - 5s 14ms/step - loss: 0.2720 - accuracy: 0.9013 - val_loss: 0.2365 - val_accuracy: 0.9133
Epoch 5/20
375/375 [==============================] - 5s 14ms/step - loss: 0.2515 - accuracy: 0.9071 - val_loss: 0.2219 - val_accuracy: 0.9174
Epoch 6/20
375/375 [==============================] - 5s 13ms/step - loss: 0.2351 - accuracy: 0.9137 - val_loss: 0.2252 - val_accuracy: 0.9140
Epoch 7/20
375/375 [==============================] - 5s 13ms/step - loss: 0.2222 - accuracy: 0.9176 - val_loss: 0.2170 - val_accuracy: 0.9182
Epoch 8/20
```

```
375/375 [==============================] - 5s 14ms/step - loss: 0.2120 - accuracy: 0.9213 - val_loss: 0.2089 - val_accuracy: 0.9227
Epoch 9/20
375/375 [==============================] - 5s 12ms/step - loss: 0.1989 - accuracy: 0.9260 - val_loss: 0.2011 - val_accuracy: 0.9265
Epoch 10/20
375/375 [==============================] - 5s 15ms/step - loss: 0.1929 - accuracy: 0.9278 - val_loss: 0.2014 - val_accuracy: 0.9268
Epoch 11/20
375/375 [==============================] - 5s 13ms/step - loss: 0.1856 - accuracy: 0.9312 - val_loss: 0.1925 - val_accuracy: 0.9302
Epoch 12/20
375/375 [==============================] - 5s 14ms/step - loss: 0.1781 - accuracy: 0.9330 - val_loss: 0.1904 - val_accuracy: 0.9321
Epoch 13/20
375/375 [==============================] - 5s 13ms/step - loss: 0.1699 - accuracy: 0.9370 - val_loss: 0.1985 - val_accuracy: 0.9302
Epoch 14/20
375/375 [==============================] - 5s 13ms/step - loss: 0.1617 - accuracy: 0.9403 - val_loss: 0.1973 - val_accuracy: 0.9303
Epoch 15/20
375/375 [==============================] - 5s 13ms/step - loss: 0.1574 - accuracy: 0.9417 - val_loss: 0.2003 - val_accuracy: 0.9302
Epoch 16/20
375/375 [==============================] - 5s 13ms/step - loss: 0.1516 - accuracy: 0.9429 - val_loss: 0.1945 - val_accuracy: 0.9327
Epoch 17/20
375/375 [==============================] - 5s 14ms/step - loss: 0.1476 - accuracy: 0.9448 - val_loss: 0.1986 - val_accuracy: 0.9302
Epoch 18/20
375/375 [==============================] - 5s 13ms/step - loss: 0.1433 - accuracy: 0.9460 - val_loss: 0.1966 - val_accuracy: 0.9307
Epoch 19/20
375/375 [==============================] - 5s 13ms/step - loss: 0.1363 - accuracy: 0.9476 - val_loss: 0.2010 - val_accuracy: 0.9315
Epoch 20/20
375/375 [==============================] - 5s 14ms/step - loss: 0.1360 - accuracy: 0.9503 - val_loss: 0.2057 - val_accuracy: 0.9283
313/313 [==============================] - 1s 3ms/step - loss: 0.2260 - accuracy: 0.9273
Test accuracy: 0.927299976348877
```

VGG on CIFAR10

```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import CategoricalCrossentropy

# Load and preprocess the CIFAR-10 dataset
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()

# Normalize the images
train_images = train_images.astype('float32') / 255
test_images = test_images.astype('float32') / 255

# Convert labels to categorical one-hot encoding
train_labels = to_categorical(train_labels, 10)
test_labels = to_categorical(test_labels, 10)

# Define the VGG-style model architecture
def VGG():
    model = Sequential([
        Conv2D(64, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)),
        Conv2D(64, (3, 3), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Conv2D(128, (3, 3), activation='relu', padding='same'),
        Conv2D(128, (3, 3), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Conv2D(256, (3, 3), activation='relu', padding='same'),
        Conv2D(256, (3, 3), activation='relu', padding='same'),
        Conv2D(256, (3, 3), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Conv2D(512, (3, 3), activation='relu', padding='same'),
        Conv2D(512, (3, 3), activation='relu', padding='same'),
        Conv2D(512, (3, 3), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Conv2D(512, (3, 3), activation='relu', padding='same'),
        Conv2D(512, (3, 3), activation='relu', padding='same'),
        Conv2D(512, (3, 3), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Flatten(),
        Dense(512, activation='relu'),
        Dropout(0.5),
```

```
        Dense(512, activation='relu'),
        Dropout(0.5),
        Dense(10, activation='softmax')
    ])
    return model

# Compile the model
model = VGG()
model.compile(optimizer=Adam(),
              loss=CategoricalCrossentropy(),
              metrics=['accuracy'])

# Train the model
history = model.fit(train_images, train_labels, epochs=50, batch_size=128, validation_split=0.2)

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc}')

# Save the results
results = {
    'test_loss': test_loss,
    'test_accuracy': test_acc,
    'history': history.history
}

import json
with open('vgg_cifar10_results.json', 'w') as f:
    json.dump(results, f)
```

```
Epoch 1/50
313/313 [==============================] - 31s 72ms/step - loss: 2.3051 - accuracy: 0.0997 - val_loss: 2.3026 - val_accuracy: 0.
Epoch 2/50
313/313 [==============================] - 19s 61ms/step - loss: 2.3032 - accuracy: 0.0981 - val_loss: 2.3027 - val_accuracy: 0.
Epoch 3/50
313/313 [==============================] - 19s 61ms/step - loss: 2.3028 - accuracy: 0.1008 - val_loss: 2.3031 - val_accuracy: 0.
Epoch 4/50
313/313 [==============================] - 19s 60ms/step - loss: 2.3028 - accuracy: 0.1004 - val_loss: 2.3029 - val_accuracy: 0.
Epoch 5/50
313/313 [==============================] - 19s 60ms/step - loss: 2.3027 - accuracy: 0.0994 - val_loss: 2.3028 - val_accuracy: 0.
Epoch 6/50
313/313 [==============================] - 19s 61ms/step - loss: 2.3028 - accuracy: 0.0982 - val_loss: 2.3028 - val_accuracy: 0.
Epoch 7/50
313/313 [==============================] - 19s 61ms/step - loss: 2.3027 - accuracy: 0.0985 - val_loss: 2.3028 - val_accuracy: 0.
Epoch 8/50
313/313 [==============================] - 19s 60ms/step - loss: 2.3027 - accuracy: 0.0986 - val_loss: 2.3027 - val_accuracy: 0.
Epoch 9/50
313/313 [==============================] - 19s 60ms/step - loss: 2.3027 - accuracy: 0.0998 - val_loss: 2.3028 - val_accuracy: 0.
Epoch 10/50
313/313 [==============================] - 19s 62ms/step - loss: 2.3027 - accuracy: 0.0988 - val_loss: 2.3027 - val_accuracy: 0.
Epoch 11/50
313/313 [==============================] - 19s 62ms/step - loss: 2.3027 - accuracy: 0.1023 - val_loss: 2.3026 - val_accuracy: 0.
Epoch 12/50
313/313 [==============================] - 19s 61ms/step - loss: 2.3027 - accuracy: 0.1006 - val_loss: 2.3027 - val_accuracy: 0.
Epoch 13/50
313/313 [==============================] - 19s 60ms/step - loss: 2.3027 - accuracy: 0.0988 - val_loss: 2.3027 - val_accuracy: 0.
Epoch 14/50
313/313 [==============================] - 19s 60ms/step - loss: 2.3027 - accuracy: 0.0983 - val_loss: 2.3026 - val_accuracy: 0.
Epoch 15/50
313/313 [==============================] - 19s 60ms/step - loss: 2.3027 - accuracy: 0.0977 - val_loss: 2.3027 - val_accuracy: 0.
Epoch 16/50
313/313 [==============================] - 19s 61ms/step - loss: 2.3027 - accuracy: 0.1005 - val_loss: 2.3027 - val_accuracy: 0.
Epoch 17/50
313/313 [==============================] - 19s 61ms/step - loss: 2.3027 - accuracy: 0.1007 - val_loss: 2.3027 - val_accuracy: 0.
Epoch 18/50
313/313 [==============================] - 19s 61ms/step - loss: 2.3027 - accuracy: 0.0983 - val_loss: 2.3027 - val_accuracy: 0.
Epoch 19/50
313/313 [==============================] - 19s 60ms/step - loss: 2.3027 - accuracy: 0.1008 - val_loss: 2.3027 - val_accuracy: 0.
Epoch 20/50
313/313 [==============================] - 19s 60ms/step - loss: 2.3027 - accuracy: 0.0990 - val_loss: 2.3028 - val_accuracy: 0.
Epoch 21/50
313/313 [==============================] - 19s 60ms/step - loss: 2.3027 - accuracy: 0.1004 - val_loss: 2.3028 - val_accuracy: 0.
Epoch 22/50
313/313 [==============================] - 19s 59ms/step - loss: 2.3027 - accuracy: 0.1015 - val_loss: 2.3028 - val_accuracy: 0.
Epoch 23/50
313/313 [==============================] - 19s 60ms/step - loss: 2.3027 - accuracy: 0.1005 - val_loss: 2.3027 - val_accuracy: 0.
Epoch 24/50
313/313 [==============================] - 19s 60ms/step - loss: 2.3027 - accuracy: 0.0965 - val_loss: 2.3027 - val_accuracy: 0.
Epoch 25/50
313/313 [==============================] - 19s 61ms/step - loss: 2.3027 - accuracy: 0.0997 - val_loss: 2.3027 - val_accuracy: 0.
Epoch 26/50
313/313 [==============================] - 19s 60ms/step - loss: 2.3027 - accuracy: 0.0989 - val_loss: 2.3028 - val_accuracy: 0.
Epoch 27/50
313/313 [==============================] - 19s 59ms/step - loss: 2.3027 - accuracy: 0.1012 - val_loss: 2.3027 - val_accuracy: 0.
Epoch 28/50
313/313 [==============================] - 19s 61ms/step - loss: 2.3027 - accuracy: 0.0961 - val_loss: 2.3028 - val_accuracy: 0.
Epoch 29/50
```

VGG on CIFAR100

```
import tensorflow as tf
from tensorflow.keras.datasets import cifar100
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import CategoricalCrossentropy

# Load and preprocess the CIFAR-100 dataset
(train_images, train_labels), (test_images, test_labels) = cifar100.load_data()

# Normalize the images
train_images = train_images.astype('float32') / 255
test_images = test_images.astype('float32') / 255

# Convert labels to categorical one-hot encoding
train_labels = to_categorical(train_labels, 100)
test_labels = to_categorical(test_labels, 100)

# Define the VGG-style model architecture
def VGG():
    model = Sequential([
        Conv2D(64, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)),
        Conv2D(64, (3, 3), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Conv2D(128, (3, 3), activation='relu', padding='same'),
        Conv2D(128, (3, 3), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Conv2D(256, (3, 3), activation='relu', padding='same'),
        Conv2D(256, (3, 3), activation='relu', padding='same'),
        Conv2D(256, (3, 3), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Conv2D(512, (3, 3), activation='relu', padding='same'),
        Conv2D(512, (3, 3), activation='relu', padding='same'),
        Conv2D(512, (3, 3), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Conv2D(512, (3, 3), activation='relu', padding='same'),
        Conv2D(512, (3, 3), activation='relu', padding='same'),
        Conv2D(512, (3, 3), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        Flatten(),
        Dense(512, activation='relu'),
        Dropout(0.5),
        Dense(512, activation='relu'),
        Dropout(0.5),
        Dense(100, activation='softmax')
    ])
    return model

# Compile the model
model = VGG()
model.compile(optimizer=Adam(),
              loss=CategoricalCrossentropy(),
              metrics=['accuracy'])

# Train the model
history = model.fit(train_images, train_labels, epochs=50, batch_size=128, validation_split=0.2)

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc}')

# Save the results
results = {
    'test_loss': test_loss,
    'test_accuracy': test_acc,
    'history': history.history
}
```

```
import json
with open('vgg_cifar100_results.json', 'w') as f:
    json.dump(results, f)
```

```
Epoch 1/50
313/313 [==============================] - 24s 62ms/step - loss: 4.6064 - accuracy: 0.0089 - val_loss: 4.6065 - val_accuracy: 0.
Epoch 2/50
313/313 [==============================] - 19s 61ms/step - loss: 4.6057 - accuracy: 0.0092 - val_loss: 4.6067 - val_accuracy: 0.
Epoch 3/50
313/313 [==============================] - 19s 60ms/step - loss: 4.6055 - accuracy: 0.0094 - val_loss: 4.6069 - val_accuracy: 0.
Epoch 4/50
313/313 [==============================] - 19s 61ms/step - loss: 4.6054 - accuracy: 0.0093 - val_loss: 4.6071 - val_accuracy: 0.
Epoch 5/50
313/313 [==============================] - 19s 61ms/step - loss: 4.6053 - accuracy: 0.0097 - val_loss: 4.6074 - val_accuracy: 0.
Epoch 6/50
313/313 [==============================] - 19s 62ms/step - loss: 4.6053 - accuracy: 0.0095 - val_loss: 4.6075 - val_accuracy: 0.
Epoch 7/50
313/313 [==============================] - 19s 61ms/step - loss: 4.6053 - accuracy: 0.0105 - val_loss: 4.6076 - val_accuracy: 0.
Epoch 8/50
313/313 [==============================] - 19s 61ms/step - loss: 4.6053 - accuracy: 0.0100 - val_loss: 4.6077 - val_accuracy: 0.
Epoch 9/50
313/313 [==============================] - 19s 61ms/step - loss: 4.6053 - accuracy: 0.0096 - val_loss: 4.6077 - val_accuracy: 0.
Epoch 10/50
313/313 [==============================] - 19s 61ms/step - loss: 4.6052 - accuracy: 0.0103 - val_loss: 4.6078 - val_accuracy: 0.
Epoch 11/50
313/313 [==============================] - 19s 61ms/step - loss: 4.6053 - accuracy: 0.0106 - val_loss: 4.6078 - val_accuracy: 0.
Epoch 12/50
313/313 [==============================] - 19s 62ms/step - loss: 4.6053 - accuracy: 0.0101 - val_loss: 4.6079 - val_accuracy: 0.
Epoch 13/50
313/313 [==============================] - 19s 61ms/step - loss: 4.6053 - accuracy: 0.0102 - val_loss: 4.6079 - val_accuracy: 0.
Epoch 14/50
313/313 [==============================] - 19s 61ms/step - loss: 4.6053 - accuracy: 0.0102 - val_loss: 4.6079 - val_accuracy: 0.
Epoch 15/50
313/313 [==============================] - 19s 61ms/step - loss: 4.6053 - accuracy: 0.0099 - val_loss: 4.6079 - val_accuracy: 0.
Epoch 16/50
313/313 [==============================] - 19s 60ms/step - loss: 4.6053 - accuracy: 0.0106 - val_loss: 4.6080 - val_accuracy: 0.
Epoch 17/50
313/313 [==============================] - 19s 61ms/step - loss: 4.6053 - accuracy: 0.0106 - val_loss: 4.6080 - val_accuracy: 0.
Epoch 18/50
313/313 [==============================] - 19s 60ms/step - loss: 4.6053 - accuracy: 0.0097 - val_loss: 4.6079 - val_accuracy: 0.
Epoch 19/50
313/313 [==============================] - 19s 61ms/step - loss: 4.6053 - accuracy: 0.0101 - val_loss: 4.6080 - val_accuracy: 0.
Epoch 20/50
313/313 [==============================] - 19s 61ms/step - loss: 4.6053 - accuracy: 0.0103 - val_loss: 4.6080 - val_accuracy: 0.
Epoch 21/50
313/313 [==============================] - 19s 61ms/step - loss: 4.6053 - accuracy: 0.0104 - val_loss: 4.6080 - val_accuracy: 0.
Epoch 22/50
313/313 [==============================] - 19s 60ms/step - loss: 4.6052 - accuracy: 0.0106 - val_loss: 4.6080 - val_accuracy: 0.
Epoch 23/50
313/313 [==============================] - 19s 60ms/step - loss: 4.6053 - accuracy: 0.0100 - val_loss: 4.6080 - val_accuracy: 0.
Epoch 24/50
313/313 [==============================] - 19s 60ms/step - loss: 4.6052 - accuracy: 0.0100 - val_loss: 4.6080 - val_accuracy: 0.
Epoch 25/50
313/313 [==============================] - 19s 60ms/step - loss: 4.6052 - accuracy: 0.0103 - val_loss: 4.6080 - val_accuracy: 0.
Epoch 26/50
313/313 [==============================] - 19s 60ms/step - loss: 4.6053 - accuracy: 0.0106 - val_loss: 4.6080 - val_accuracy: 0.
Epoch 27/50
313/313 [==============================] - 19s 61ms/step - loss: 4.6053 - accuracy: 0.0102 - val_loss: 4.6080 - val_accuracy: 0.
Epoch 28/50
313/313 [==============================] - 19s 60ms/step - loss: 4.6053 - accuracy: 0.0106 - val_loss: 4.6079 - val_accuracy: 0.
Epoch 29/50
```

ResNet on MNIST

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, BatchNormalization, Activation, Add, Flatten, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import CategoricalCrossentropy

# Load and preprocess the MNIST dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Reshape and normalize the images
train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255

# Convert labels to categorical one-hot encoding
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# Define the ResNet model architecture
def resnet_block(input_tensor, filters, kernel_size=3, stride=1):
    x = Conv2D(filters, kernel_size=kernel_size, strides=stride, padding='same')(input_tensor)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    x = Conv2D(filters, kernel_size=kernel_size, strides=1, padding='same')(x)
    x = BatchNormalization()(x)

    if stride != 1:
        input_tensor = Conv2D(filters, kernel_size=1, strides=stride, padding='same')(input_tensor)
        input_tensor = BatchNormalization()(input_tensor)

    x = Add()([x, input_tensor])
    x = Activation('relu')(x)
    return x

def ResNet(input_shape, num_classes):
    inputs = Input(shape=input_shape)
    x = Conv2D(32, kernel_size=3, strides=1, padding='same')(inputs)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    x = resnet_block(x, 32)
    x = resnet_block(x, 32)
    x = resnet_block(x, 64, stride=2)
    x = resnet_block(x, 64)
    x = resnet_block(x, 128, stride=2)
    x = resnet_block(x, 128)

    x = Flatten()(x)
    x = Dense(256, activation='relu')(x)
    x = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs, x)
    return model

# Compile the model
model = ResNet(input_shape=(28, 28, 1), num_classes=10)
model.compile(optimizer=Adam(),
              loss=CategoricalCrossentropy(),
              metrics=['accuracy'])

# Train the model
history = model.fit(train_images, train_labels, epochs=20, batch_size=128, validation_split=0.2)

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc}')

# Save the results
results = {
    'test_loss': test_loss,
    'test_accuracy': test_acc,
    'history': history.history
}

import json
with open('resnet_mnist_results.json', 'w') as f:
    json.dump(results, f)
```

```
Epoch 1/20
375/375 [==============================] - 25s 42ms/step - loss: 0.2934 - accuracy: 0.9395 - val_loss: 1.5562 - val_accuracy: 0.506
Epoch 2/20
375/375 [==============================] - 14s 37ms/step - loss: 0.0461 - accuracy: 0.9860 - val_loss: 0.0823 - val_accuracy: 0.976
Epoch 3/20
375/375 [==============================] - 14s 37ms/step - loss: 0.0364 - accuracy: 0.9889 - val_loss: 0.0759 - val_accuracy: 0.980
Epoch 4/20
375/375 [==============================] - 13s 36ms/step - loss: 0.0241 - accuracy: 0.9924 - val_loss: 0.0484 - val_accuracy: 0.986
Epoch 5/20
375/375 [==============================] - 14s 36ms/step - loss: 0.0217 - accuracy: 0.9929 - val_loss: 0.0560 - val_accuracy: 0.986
Epoch 6/20
375/375 [==============================] - 13s 35ms/step - loss: 0.0230 - accuracy: 0.9923 - val_loss: 0.0618 - val_accuracy: 0.983
Epoch 7/20
375/375 [==============================] - 14s 37ms/step - loss: 0.0197 - accuracy: 0.9938 - val_loss: 0.0457 - val_accuracy: 0.987
Epoch 8/20
375/375 [==============================] - 14s 36ms/step - loss: 0.0152 - accuracy: 0.9950 - val_loss: 0.0753 - val_accuracy: 0.981
Epoch 9/20
375/375 [==============================] - 14s 36ms/step - loss: 0.0175 - accuracy: 0.9949 - val_loss: 0.0570 - val_accuracy: 0.986
Epoch 10/20
375/375 [==============================] - 14s 37ms/step - loss: 0.0107 - accuracy: 0.9963 - val_loss: 0.0808 - val_accuracy: 0.979
Epoch 11/20
375/375 [==============================] - 13s 36ms/step - loss: 0.0144 - accuracy: 0.9954 - val_loss: 0.0819 - val_accuracy: 0.980
Epoch 12/20
375/375 [==============================] - 13s 36ms/step - loss: 0.0112 - accuracy: 0.9964 - val_loss: 0.0545 - val_accuracy: 0.987
Epoch 13/20
375/375 [==============================] - 14s 36ms/step - loss: 0.0130 - accuracy: 0.9958 - val_loss: 0.0448 - val_accuracy: 0.990
Epoch 14/20
375/375 [==============================] - 14s 36ms/step - loss: 0.0091 - accuracy: 0.9969 - val_loss: 0.0390 - val_accuracy: 0.989
Epoch 15/20
375/375 [==============================] - 14s 37ms/step - loss: 0.0131 - accuracy: 0.9960 - val_loss: 0.0507 - val_accuracy: 0.989
Epoch 16/20
375/375 [==============================] - 14s 37ms/step - loss: 0.0080 - accuracy: 0.9975 - val_loss: 0.0550 - val_accuracy: 0.987
Epoch 17/20
375/375 [==============================] - 14s 36ms/step - loss: 0.0069 - accuracy: 0.9978 - val_loss: 0.0512 - val_accuracy: 0.988
Epoch 18/20
375/375 [==============================] - 14s 37ms/step - loss: 0.0071 - accuracy: 0.9978 - val_loss: 0.0405 - val_accuracy: 0.991
Epoch 19/20
375/375 [==============================] - 14s 37ms/step - loss: 0.0104 - accuracy: 0.9966 - val_loss: 0.0408 - val_accuracy: 0.989
Epoch 20/20
375/375 [==============================] - 14s 37ms/step - loss: 0.0043 - accuracy: 0.9987 - val_loss: 0.0522 - val_accuracy: 0.990
313/313 [==============================] - 2s 6ms/step - loss: 0.0344 - accuracy: 0.9923
Test accuracy: 0.9922999739646912
```

ResNet on Fashion-MNIST

```python
import tensorflow as tf
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, BatchNormalization, Activation, Add, Flatten, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import CategoricalCrossentropy

# Load and preprocess the Fashion MNIST dataset
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

# Reshape and normalize the images
train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255

# Convert labels to categorical one-hot encoding
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# Define the ResNet model architecture
def resnet_block(input_tensor, filters, kernel_size=3, stride=1):
    x = Conv2D(filters, kernel_size=kernel_size, strides=stride, padding='same')(input_tensor)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    x = Conv2D(filters, kernel_size=kernel_size, strides=1, padding='same')(x)
    x = BatchNormalization()(x)

    if stride != 1:
        input_tensor = Conv2D(filters, kernel_size=1, strides=stride, padding='same')(input_tensor)
        input_tensor = BatchNormalization()(input_tensor)

    x = Add()([x, input_tensor])
    x = Activation('relu')(x)
    return x

def ResNet(input_shape, num_classes):
    inputs = Input(shape=input_shape)
    x = Conv2D(32, kernel_size=3, strides=1, padding='same')(inputs)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    x = resnet_block(x, 32)
    x = resnet_block(x, 32)
    x = resnet_block(x, 64, stride=2)
    x = resnet_block(x, 64)
    x = resnet_block(x, 128, stride=2)
    x = resnet_block(x, 128)

    x = Flatten()(x)
    x = Dense(256, activation='relu')(x)
    x = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs, x)
    return model

# Compile the model
model = ResNet(input_shape=(28, 28, 1), num_classes=10)
model.compile(optimizer=Adam(),
              loss=CategoricalCrossentropy(),
              metrics=['accuracy'])

# Train the model
history = model.fit(train_images, train_labels, epochs=20, batch_size=128, validation_split=0.2)

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc}')

# Save the results
results = {
    'test_loss': test_loss,
    'test_accuracy': test_acc,
    'history': history.history
}

import json
with open('resnet_fashion_mnist_results.json', 'w') as f:
    json.dump(results, f)
```

```
Epoch 1/20
375/375 [==============================] - 23s 37ms/step - loss: 0.5958 - accuracy: 0.8256 - val_loss: 1.0386 - val_accuracy: 0.662
Epoch 2/20
375/375 [==============================] - 14s 36ms/step - loss: 0.2634 - accuracy: 0.9045 - val_loss: 0.2647 - val_accuracy: 0.908
Epoch 3/20
375/375 [==============================] - 14s 36ms/step - loss: 0.2064 - accuracy: 0.9246 - val_loss: 0.2668 - val_accuracy: 0.902
Epoch 4/20
375/375 [==============================] - 14s 37ms/step - loss: 0.1720 - accuracy: 0.9357 - val_loss: 0.2598 - val_accuracy: 0.910
Epoch 5/20
375/375 [==============================] - 14s 37ms/step - loss: 0.1416 - accuracy: 0.9474 - val_loss: 0.3288 - val_accuracy: 0.895
Epoch 6/20
375/375 [==============================] - 13s 36ms/step - loss: 0.1174 - accuracy: 0.9566 - val_loss: 0.2469 - val_accuracy: 0.915
Epoch 7/20
375/375 [==============================] - 14s 36ms/step - loss: 0.1001 - accuracy: 0.9637 - val_loss: 0.2487 - val_accuracy: 0.920
Epoch 8/20
375/375 [==============================] - 14s 36ms/step - loss: 0.0793 - accuracy: 0.9707 - val_loss: 0.2499 - val_accuracy: 0.917
Epoch 9/20
375/375 [==============================] - 14s 37ms/step - loss: 0.0634 - accuracy: 0.9770 - val_loss: 0.3024 - val_accuracy: 0.915
Epoch 10/20
375/375 [==============================] - 14s 37ms/step - loss: 0.0523 - accuracy: 0.9802 - val_loss: 0.3105 - val_accuracy: 0.916
Epoch 11/20
375/375 [==============================] - 14s 36ms/step - loss: 0.0482 - accuracy: 0.9823 - val_loss: 0.3611 - val_accuracy: 0.915
Epoch 12/20
375/375 [==============================] - 13s 36ms/step - loss: 0.0370 - accuracy: 0.9867 - val_loss: 0.3547 - val_accuracy: 0.919
Epoch 13/20
375/375 [==============================] - 13s 36ms/step - loss: 0.0374 - accuracy: 0.9863 - val_loss: 0.3962 - val_accuracy: 0.911
Epoch 14/20
375/375 [==============================] - 13s 36ms/step - loss: 0.0322 - accuracy: 0.9883 - val_loss: 0.3440 - val_accuracy: 0.923
Epoch 15/20
375/375 [==============================] - 13s 35ms/step - loss: 0.0284 - accuracy: 0.9900 - val_loss: 0.5480 - val_accuracy: 0.902
Epoch 16/20
375/375 [==============================] - 14s 37ms/step - loss: 0.0265 - accuracy: 0.9906 - val_loss: 0.3532 - val_accuracy: 0.925
Epoch 17/20
375/375 [==============================] - 14s 37ms/step - loss: 0.0210 - accuracy: 0.9926 - val_loss: 0.3818 - val_accuracy: 0.918
Epoch 18/20
375/375 [==============================] - 14s 37ms/step - loss: 0.0203 - accuracy: 0.9927 - val_loss: 0.4245 - val_accuracy: 0.919
Epoch 19/20
375/375 [==============================] - 14s 37ms/step - loss: 0.0495 - accuracy: 0.9835 - val_loss: 0.3742 - val_accuracy: 0.919
Epoch 20/20
375/375 [==============================] - 14s 37ms/step - loss: 0.0144 - accuracy: 0.9950 - val_loss: 0.4456 - val_accuracy: 0.917
313/313 [==============================] - 2s 5ms/step - loss: 0.4558 - accuracy: 0.9166
Test accuracy: 0.9165999889373779
```

ResNet on CIFAR10

```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, Input
from tensorflow.keras.optimizers import Adam

# Set random seed for reproducibility
tf.random.set_seed(42)

# Load and preprocess the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

def create_resnet_model(input_shape, num_classes):
    base_model = ResNet50(include_top=False, input_shape=input_shape, pooling='avg')
    base_model.trainable = False
    inputs = Input(shape=input_shape)
    x = base_model(inputs, training=False)
    x = Flatten()(x)
    outputs = Dense(num_classes, activation='softmax')(x)
    model = Model(inputs, outputs)
    return model

input_shape = (32, 32, 3)
num_classes = 10
model = create_resnet_model(input_shape, num_classes)

# Compile the model
model.compile(optimizer=Adam(),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(x_train, y_train,
                    epochs=20,
                    batch_size=64,
                    validation_split=0.1,
                    verbose=2)

# Evaluate the model
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=0)
print(f'Test accuracy: {test_accuracy:.4f}')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/re
94765736/94765736 [==============================] - 5s 0us/step
Epoch 1/20
704/704 - 16s - loss: 2.1209 - accuracy: 0.2269 - val_loss: 2.0164 - val_accuracy: 0.
Epoch 2/20
704/704 - 10s - loss: 1.9570 - accuracy: 0.2969 - val_loss: 1.9081 - val_accuracy: 0.
Epoch 3/20
704/704 - 9s - loss: 1.8948 - accuracy: 0.3184 - val_loss: 1.8656 - val_accuracy: 0.3
Epoch 4/20
704/704 - 9s - loss: 1.8666 - accuracy: 0.3336 - val_loss: 1.8757 - val_accuracy: 0.3
Epoch 5/20
704/704 - 9s - loss: 1.8373 - accuracy: 0.3462 - val_loss: 1.8329 - val_accuracy: 0.3
Epoch 6/20
704/704 - 8s - loss: 1.8128 - accuracy: 0.3547 - val_loss: 1.7989 - val_accuracy: 0.3
Epoch 7/20
704/704 - 9s - loss: 1.7946 - accuracy: 0.3634 - val_loss: 1.8366 - val_accuracy: 0.3
Epoch 8/20
704/704 - 8s - loss: 1.7814 - accuracy: 0.3677 - val_loss: 1.7637 - val_accuracy: 0.3
Epoch 9/20
704/704 - 9s - loss: 1.7736 - accuracy: 0.3712 - val_loss: 1.8165 - val_accuracy: 0.3
Epoch 10/20
704/704 - 9s - loss: 1.7655 - accuracy: 0.3740 - val_loss: 1.7315 - val_accuracy: 0.4
Epoch 11/20
704/704 - 8s - loss: 1.7523 - accuracy: 0.3770 - val_loss: 1.7291 - val_accuracy: 0.3
Epoch 12/20
704/704 - 9s - loss: 1.7425 - accuracy: 0.3836 - val_loss: 1.7147 - val_accuracy: 0.4
Epoch 13/20
704/704 - 9s - loss: 1.7356 - accuracy: 0.3852 - val_loss: 1.7084 - val_accuracy: 0.4
Epoch 14/20
704/704 - 9s - loss: 1.7277 - accuracy: 0.3886 - val_loss: 1.7004 - val_accuracy: 0.4
Epoch 15/20
704/704 - 8s - loss: 1.7194 - accuracy: 0.3922 - val_loss: 1.7219 - val_accuracy: 0.3
Epoch 16/20
704/704 - 9s - loss: 1.7130 - accuracy: 0.3938 - val_loss: 1.6996 - val_accuracy: 0.3
Epoch 17/20
704/704 - 9s - loss: 1.7055 - accuracy: 0.3974 - val_loss: 1.6911 - val_accuracy: 0.4
Epoch 18/20
704/704 - 9s - loss: 1.7026 - accuracy: 0.4009 - val_loss: 1.6952 - val_accuracy: 0.3
Epoch 19/20
704/704 - 9s - loss: 1.6970 - accuracy: 0.4002 - val_loss: 1.6654 - val_accuracy: 0.4
Epoch 20/20
704/704 - 10s - loss: 1.6924 - accuracy: 0.4009 - val_loss: 1.6994 - val_accuracy: 0.
Test accuracy: 0.3869
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-12-8fdd2e2dcb2e> in <cell line: 50>()
     48 
     49 # Fine-tuning (optional)
---> 50 base_model.trainable = True
     51 model.compile(optimizer=Adam(1e-5),
     52               loss='categorical_crossentropy',

NameError: name 'base_model' is not defined
```

Next steps: Explain error

ResNet on CIFAR100

```
import tensorflow as tf
from tensorflow.keras.datasets import cifar100
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, Input
from tensorflow.keras.optimizers import Adam

# Set random seed for reproducibility
tf.random.set_seed(42)

# Load and preprocess the CIFAR-100 dataset
(x_train, y_train), (x_test, y_test) = cifar100.load_data()
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
y_train = to_categorical(y_train, 100)
y_test = to_categorical(y_test, 100)

def create_resnet_model(input_shape, num_classes):
    base_model = ResNet50(include_top=False, input_shape=input_shape, pooling='avg')
    base_model.trainable = False
    inputs = Input(shape=input_shape)
    x = base_model(inputs, training=False)
    x = Flatten()(x)
    outputs = Dense(num_classes, activation='softmax')(x)
    model = Model(inputs, outputs)
    return model

input_shape = (32, 32, 3)
num_classes = 100
model = create_resnet_model(input_shape, num_classes)

# Compile the model
model.compile(optimizer=Adam(),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(x_train, y_train,
                    epochs=20,
                    batch_size=64,
                    validation_split=0.1,
                    verbose=2)

# Evaluate the model
test_loss, test_accuracy = model.evaluate(x_test, y_test, verbose=0)
print(f'Test accuracy: {test_accuracy:.4f}')
```

```
Epoch 1/20
704/704 - 14s - loss: 4.5818 - accuracy: 0.0290 - val_loss: 4.4901 - val_accuracy: 0.0370 - 14s/epoch - 20ms/step
Epoch 2/20
704/704 - 9s - loss: 4.3770 - accuracy: 0.0529 - val_loss: 4.2870 - val_accuracy: 0.0672 - 9s/epoch - 13ms/step
Epoch 3/20
704/704 - 10s - loss: 4.2779 - accuracy: 0.0685 - val_loss: 4.2433 - val_accuracy: 0.0734 - 10s/epoch - 15ms/step
Epoch 4/20
704/704 - 9s - loss: 4.2067 - accuracy: 0.0790 - val_loss: 4.2418 - val_accuracy: 0.0718 - 9s/epoch - 13ms/step
Epoch 5/20
704/704 - 9s - loss: 4.1614 - accuracy: 0.0847 - val_loss: 4.1889 - val_accuracy: 0.0890 - 9s/epoch - 13ms/step
Epoch 6/20
704/704 - 9s - loss: 4.1266 - accuracy: 0.0905 - val_loss: 4.1557 - val_accuracy: 0.0830 - 9s/epoch - 12ms/step
Epoch 7/20
704/704 - 9s - loss: 4.0906 - accuracy: 0.0951 - val_loss: 4.1185 - val_accuracy: 0.0860 - 9s/epoch - 13ms/step
Epoch 8/20
704/704 - 9s - loss: 4.0594 - accuracy: 0.0983 - val_loss: 4.0431 - val_accuracy: 0.0972 - 9s/epoch - 12ms/step
Epoch 9/20
704/704 - 9s - loss: 4.0361 - accuracy: 0.1036 - val_loss: 4.1219 - val_accuracy: 0.0904 - 9s/epoch - 13ms/step
Epoch 10/20
704/704 - 9s - loss: 4.0152 - accuracy: 0.1066 - val_loss: 4.0919 - val_accuracy: 0.0988 - 9s/epoch - 12ms/step
Epoch 11/20
704/704 - 9s - loss: 3.9912 - accuracy: 0.1100 - val_loss: 4.0962 - val_accuracy: 0.1024 - 9s/epoch - 13ms/step
Epoch 12/20
704/704 - 9s - loss: 3.9765 - accuracy: 0.1118 - val_loss: 4.0653 - val_accuracy: 0.0960 - 9s/epoch - 13ms/step
Epoch 13/20
704/704 - 9s - loss: 3.9522 - accuracy: 0.1156 - val_loss: 4.0056 - val_accuracy: 0.1160 - 9s/epoch - 12ms/step
Epoch 14/20
704/704 - 9s - loss: 3.9348 - accuracy: 0.1186 - val_loss: 3.9858 - val_accuracy: 0.1052 - 9s/epoch - 12ms/step
Epoch 15/20
704/704 - 9s - loss: 3.9277 - accuracy: 0.1205 - val_loss: 4.0195 - val_accuracy: 0.1068 - 9s/epoch - 13ms/step
Epoch 16/20
704/704 - 9s - loss: 3.9124 - accuracy: 0.1225 - val_loss: 3.9551 - val_accuracy: 0.1134 - 9s/epoch - 13ms/step
Epoch 17/20
704/704 - 9s - loss: 3.8983 - accuracy: 0.1239 - val_loss: 3.9776 - val_accuracy: 0.1150 - 9s/epoch - 13ms/step
```

```
Epoch 18/20
704/704 - 9s - loss: 3.8830 - accuracy: 0.1266 - val_loss: 3.9820 - val_accuracy: 0.1152 - 9s/epoch - 13ms/step
Epoch 19/20
704/704 - 8s - loss: 3.8677 - accuracy: 0.1284 - val_loss: 3.9198 - val_accuracy: 0.1208 - 8s/epoch - 12ms/step
Epoch 20/20
704/704 - 8s - loss: 3.8576 - accuracy: 0.1319 - val_loss: 3.9610 - val_accuracy: 0.1128 - 8s/epoch - 12ms/step
Test accuracy: 0.1140
```

DenseNet on MNIST

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Load and preprocess the MNIST dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# Define a Dense Block
def dense_block(x, blocks, name):
    for i in range(blocks):
        x = conv_block(x, 32, name=name + '_block' + str(i + 1))
    return x

# Define a Convolution Block
def conv_block(x, growth_rate, name):
    x1 = layers.BatchNormalization(name=name + '_bn')(x)
    x1 = layers.Activation('relu', name=name + '_relu')(x1)
    x1 = layers.Conv2D(4 * growth_rate, 1, use_bias=False, name=name + '_conv1')(x1)
    x1 = layers.BatchNormalization(name=name + '_bn2')(x1)
    x1 = layers.Activation('relu', name=name + '_relu2')(x1)
    x1 = layers.Conv2D(growth_rate, 3, padding='same', use_bias=False, name=name + '_conv2')(x1)
    x = layers.Concatenate(name=name + '_concat')([x, x1])
    return x

# Define a Transition Layer
def transition_block(x, reduction, name):
    x = layers.BatchNormalization(name=name + '_bn')(x)
    x = layers.Activation('relu', name=name + '_relu')(x)
    x = layers.Conv2D(int(tf.keras.backend.int_shape(x)[-1] * reduction), 1, use_bias=False, name=name + '_conv')(x)
    x = layers.AveragePooling2D(2, strides=2, name=name + '_pool')(x)
    return x

# Define the DenseNet model
def DenseNet(input_shape, num_classes):
    inputs = layers.Input(shape=input_shape)

    x = layers.Conv2D(64, 3, padding='same', use_bias=False, name='conv1/conv')(inputs)
    x = layers.BatchNormalization(name='conv1/bn')(x)
    x = layers.Activation('relu', name='conv1/relu')(x)
    x = layers.MaxPooling2D(2, strides=2, padding='same', name='pool1')(x)

    x = dense_block(x, 6, name='conv2')
    x = transition_block(x, 0.5, name='pool2')

    x = dense_block(x, 12, name='conv3')
    x = transition_block(x, 0.5, name='pool3')

    x = dense_block(x, 24, name='conv4')

    x = layers.BatchNormalization(name='bn')(x)
    x = layers.Activation('relu', name='relu')(x)
    x = layers.GlobalAveragePooling2D(name='avg_pool')(x)
    x = layers.Dense(num_classes, activation='softmax', name='fc')(x)

    model = models.Model(inputs, x, name='densenet')
    return model

# Build and compile the model
model = DenseNet(input_shape=(28, 28, 1), num_classes=10)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(train_images, train_labels, epochs=10, batch_size=64, validation_split=0.2)

# Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```

```
Epoch 1/10
750/750 [==============================] - 97s 68ms/step - loss: 0.0978 - accuracy: 0.9705 - val_loss: 0.0545 - val_accuracy: 0.982
Epoch 2/10
750/750 [==============================] - 51s 67ms/step - loss: 0.0422 - accuracy: 0.9872 - val_loss: 0.0435 - val_accuracy: 0.988
Epoch 3/10
750/750 [==============================] - 50s 66ms/step - loss: 0.0318 - accuracy: 0.9903 - val_loss: 0.0773 - val_accuracy: 0.979
Epoch 4/10
750/750 [==============================] - 48s 64ms/step - loss: 0.0283 - accuracy: 0.9910 - val_loss: 0.0712 - val_accuracy: 0.978
Epoch 5/10
```

```
750/750 [==============================] - 51s 68ms/step - loss: 0.0220 - accuracy: 0.9928 - val_loss: 0.0368 - val_accuracy: 0.990
```

DenseNet on Fashion-MNIST

```
Epoch 7/10
```

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.utils import to_categorical

# Load and preprocess the Fashion-MNIST dataset
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```